Universidad de las Ciencias Informáticas

Facultad 4

Núcleo de aplicaciones para el Diseño Paramétrico Asistido por Computadora en dos y tres dimensiones

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Ernesto Alejandro Santana Hidalgo

Tutores:

Dr.C Augusto Cesar Rodríguez Medina

Ing. Sandy García Santos

Declaración de autoría

Declaro ser el único autor del presente trabajo de diploma y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2017.

_____

Ernesto Alejandro Santana Hidalgo

Firma del Autor

_____ _____

Dr.C Augusto Cesar Rodríguez Medina

Firma del Tutor

Ing. Sandy García Santos

Firma del Tutor

Dedicatoria:

A mis padres, hermana y sobrinas por ser el pilar fundamental en todo lo que soy, en toda mi educación, tanto académica, como de la vida, por su incondicional apoyo.

Este trabajo ha sido posible gracias a ellos.

Agradecimientos:

Resumen

El presente trabajo contiene los resultados generales de un proceso de desarrollo de software, destinado a obtener un núcleo para aplicaciones informáticas de diseño asistido por computadora con las funcionalidades requeridas; las decisiones que definieron el resultado estuvieron fundamentadas en una investigación previa, en la que se recopiló y procesó información acerca de las tecnologías para la visualización, el estudio de los requisitos, las estructuras de datos, así como la persistencia y serialización de los datos. Una de las características proyectadas para el núcleo desde el inicio del proceso fue la capacidad de integrar módulos informáticos para el modelado de piezas que previamente se habían obtenido. Los criterios metodológicos seguidos a lo largo del proceso fueron: transitar desde los problemas más simples a los más complejos y reutilizar concepciones y funcionalidades existentes en tecnologías de código abierto disponibles.

Palabras claves: diseño asistido por computadora, módulos, núcleo.

Índice

Introducción

El presente trabajo posee sus bases en un proyecto del Grupo de Investigación “Soluciones Informáticas para la Ingeniería y la Industria” (SIPII) perteneciente al Departamento de Ciencias Básicas de la facultad 4, en la Universidad de las Ciencias Informáticas; el cuerpo del documento contiene los resultados del proceso de investigación y desarrollo, asociado al desarrollo de una aplicación que destinada a emplearse como **núcleo** en sistemas para el diseño asistido por computadora.

La idea del mencionado desarrollo surgió de la necesidad de integrar módulos obtenidos previamente, para conformar un sistema informático que permita resolver algunos de los problemas que enfrenta la industria nacional en el área de las tecnologías para el diseño asistido por computadora, asociados a las restricciones del bloqueo económico y comercial impuesto por el gobierno de los Estados Unidos de América contra Cuba.

El núcleo, como componente base de un sistema para el diseño asistido por computadora, se concibió para agrupar las funcionalidades generales de la aplicación, partiendo de un criterio arquitectónico que garantizara robustez, eficacia y posibilidad de integrar módulos con facilidad y de forma incremental.

Para garantizar niveles de independencia y soberanía en el producto previsto, se utilizaron tecnologías de código abierto como la versión comunitaria de Open CASCADE, la implementación del estándar Open Inventor Coin3D, el marco de trabajo Qt y el código fuente de la aplicación FreeCAD; se ejecutó el proceso transitando desde los aspectos más simples a los más complejos haciendo un empleo sistemático, en lo fundamental, del método teórico de análisis y síntesis y el empírico de observación.

En el proceso de recopilación de la información, que sería necesario procesar para fundamentar el trabajo, se consultaron numerosas fuentes electrónicas e impresas; particular importancia tuvo la observación de animaciones, que muestran sistemas de diseño asistido por computadoras en ejecución.

El documento está estructurado en tres capítulos; en el primero se exponen los aspectos generales de la fundamentación teórica del trabajo, iniciando con la conformación del perfil de la investigación; se incluyen además los aspectos estudiados sobre los requerimientos de los núcleos de aplicaciones para el diseño asistido por computadora, las estructuras de datos empleadas en ese tipo de aplicaciones y las tecnologías de visualización entre otros. En el segundo capítulo se hace la propuesta de solución, para lo que se parte del modelo del dominio, se exponen los requisitos capturados, se realiza la descripción del núcleo, se define la arquitectura y se establecen los patrones de diseño. En el tercero se presentan los resultados más importantes

de las etapas de implementación y de pruebas. Se concluye el trabajo relacionando las conclusiones y recomendaciones que durante el proceso se fueron obteniendo.

1 Fundamentación Teórica

En este capítulo se exponen los resultados de la investigación sobre el estado del arte en las tecnologías para el **diseño asistido por computadoras**, término que es más conocido por la denominación en inglés "**Computer Aided Design**" o "**CAD**" por sus siglas; la parte principal que se aborda es la de los núcleos de las mismas y el análisis de las funcionalidades básicas que deben poseer para garantizar la eficacia de la aplicación.

Las funcionalidades básicas de un núcleo para el tipo de sistema en desarrollo están asociadas con el empleo de las tecnologías que las encapsulan, pero el desconocimiento que en el contexto del proyecto existía sobre las formas de hacerlo impusieron la necesidad de realizar un proceso de investigación, que implicó una búsqueda exhaustiva de información en fuentes bibliográficas, códigos fuentes disponibles y la observación del funcionamiento en aplicaciones comerciales (Bayerischer Rundfunk 2016; cadpointdirect 2017; DIE TECHNICA 2017; howENGINEERSdoit! 2013) y de código abierto.

Fue necesario en un principio, para poder ubicar el problema, reunir información general sobre las tecnologías para el Diseño Asistido por Computadoras; ese fue el punto de partida para entender que entre los tópicos a estudiar se debían considerar además los procedimientos para la persistencia y serialización de los datos, las estructuras de datos para los sistemas CAD, las tecnologías para la visualización de construcciones geométricas en dos dimensiones y modelos en el espacio tridimensional, así como los fundamentos sobre los diseños arquitectónicos, siendo este último el más importante pues del éxito en la concepción de estos dependerán la eficacia del sistema y el soporte de un desarrollo incremental. En los apartados siguientes se profundiza en estos temas iniciando por algunos de los aspectos que motivaron la investigación.

1.1 Motivación del trabajo y diseño del perfil de la investigación

La motivación para el presente trabajo estuvo condicionada por necesidades de proyecto asociadas a la situación de las tecnologías de Diseño Asistido por Computadoras en el país, en los siguientes párrafos se exponen aspectos sobre el tema.

Para la ingeniería en general, el diseño asistido por computadora proporciona herramientas que automatizan los procesos de modelado, visualización y elaboración de la documentación técnica de los objetos de diseño, logrando incrementar la calidad, la eficiencia, reducir costos y acortar los tiempos de diseño y de producción.

Los sistemas CAD se consideran tecnologías de avanzada con un peso importante en la automatización de los procesos productivos; la producción de estos en las últimas décadas se ha

realizado por compañías que han estado compitiendo entre sí por el mercado con productos como AutoCAD e Inventor de AutoDesk Incorporated, SolidWorks y CATIA de Dassault Systèmes (CATIA 2017; Dassault Systèmes 2017) y SolidEdge de Siemens PLM Software, entre otros; estos sistemas implementan algoritmos para el modelado, la visualización, la edición y el manejo de los datos con niveles de eficiencia que han ido creciendo con el tiempo.

Los sistemas referidos son comerciales y Cuba no puede acceder a los mismos por las siguientes razones:

1. El bloqueo económico impuesto por el gobierno de los Estado Unidos de América a nuestro país contiene cláusulas que impiden el acceso a sistemas considerados de alta tecnología como es el caso de las tecnologías CAD (Montano 2015).
2. Son sistemas costosos y su adquisición, en caso de no existir el bloqueo, solamente estaría justificada si la comercialización de los productos resultantes asegura la recuperación de la inversión (Martínez 2004).
3. Se distribuyen, principalmente, bajo licencias de uso, lo que significa que el sistema informático no se adquiere como propiedad, sino que se autoriza su uso por un tiempo determinado.
4. A los resultados de diseño en ingeniería, como obras del intelecto humano, le son aplicables las regulaciones sobre la propiedad intelectual, por lo que si fueron obtenidas con herramientas sin licencia no es posible su comercialización (WIPO 2017).

Aunque todos los países no tienen las mismas limitaciones que Cuba, se puede apreciar que incluso algunos con fuerte desarrollo, han implementado proyectos con tecnologías de código abierto, como alternativa para enfrentar las restricciones que impone la adquisición de los sistemas comerciales; algunos de estos son los proyectos ELMER, Code-Aster, Code-Saturne y Salome-Meca, entre otros (Elmer 2017; Code-Aster 2017; Code-Saturne 2017); este hecho condujo a la idea de profundizar en el análisis de sistemas que en software libre, tuviesen la mayor cantidad de nexos con las tecnologías CAD; los estudios preliminares permitieron identificar que en esa situación se encontraban las aplicaciones FreeCAD y Salome-Meca, ambas con funcionalidades para el modelado y el diseño asistido por computadoras, distribuidas bajo licencias LGPL[1], lo que significa que se puede tener acceso a sus códigos, modificarlos y estudiarlos; también se evaluó que como elemento a favor está la disponibilidad de la tecnología Open Cascade como software libre; en las líneas que siguen se ofrece una breve descripción de las mismas.

1 GNU Lesser General Public License

FreeCAD

FreeCAD es una aplicación para el modelado en tres dimensiones basado en la tecnología Open CASCADE, con funcionalidades para el dibujo paramétrico en dos dimensiones y el diseño basado en el historial de rasgos de la pieza ("History Based Feature"); se pretende convertirlo en una plataforma para el diseño asistido por computadoras de amplio espectro, lo que permitirá modelar piezas y estructuras desde la concepción de diferentes ramas de la ingeniería como la mecánica, naval, civil o mecatrónica; en su estado actual es posible modelar prácticamente cualquier objeto del mundo real, aunque no tiene el desarrollo suficiente para las operaciones de ensamble.

El modelado paramétrico implica el empleo de parámetros durante la conformación de los modelos, con lo que es posible realizar modificaciones y actualizaciones posteriores en función de las necesidades de un proyecto, según (Bettig y Hoffmann 2011) se realiza por dos vías fundamentales: mediante la determinación de las condiciones que satisfacen ciertas restricciones geométricas ("Geometric Contraints Solving" (Brüderlin y Roller 2012)) y mediante operaciones paramétricas (altura de extrusión, ángulo de una revolución, dimensiones de las primitivas, etc.); con esta forma de hacer la configuración de los objetos 3D está controlada por parámetros, por ejemplo, la forma de un ortoedro puede ser controlada por tres parámetros: altura, ancho y longitud, en FreeCAD, al igual que en otros modeladores, estos valores que constituyen parámetros, definen la forma del objeto y pueden ser modificados en cualquier momento posterior a su creación (FreeCAD 2016a).

Salome-Meca

SALOME-Meca es un software de código abierto que proporciona una plataforma genérica para facilitar el pre y postprocesamiento de simulaciones numéricas, integra diferentes programas como Gmesh, Eficas, Code-Aster y ParaView; tiene una arquitectura abierta, flexible y es multiplataforma. Contiene un módulo denominado *Geometry* destinado al modelado geométrico con funciones similares a las de otros sistemas CAD, pero no funciona con el concepto de parametrización que de ordinario se aprecia en los sistemas comerciales y en FreeCAD; tampoco implementa el "History Based Feature"; la implementación de las funcionalidades del módulo *Geometry* está basado en la tecnología Open CASCADE.

Open CASCADE Technology

Es el único “Kernel” gráfico de código abierto destinado al desarrollo de sistemas CAD/CAE[2] y tuvo su origen en el núcleo matemático del sistema CAD EUCLID, desarrollado por la

2 Ingeniería asistida por computadora (CAE, del inglés Computer Aided Engineering)

compañía Matra Datavision Company a inicios de la década de 1980; su denominación CASCADE es el acrónimo de *Computer Aided Software for Computer Aided Design and Engineering*, derivado de la plataforma CAS.CADE liberada en el año 1993, que serviría de base a un nuevo desarrollo denominado EUCLID QUANTUM, liberado en 1996; en 1999 se decidió liberar el código con el nombre de Open CASCADE; en el 2000 se creó la filial de Matra "Open CASCADE S.A.S"; en 2003 Matra Datavision Company fue adquirida por IBM y Open CASCADE por la compañía francesa "Principia". Uno de los componentes importantes que ofrece Open CASCADE Techcnology es "*Open CASCADE Application Framework*" (OCAF) sobre el que se profundiza en el apartado 1.4.1 (Slyadnev 2014; OPEN CASCADE S.A.S 2017).

Tomando en cuenta la infraestructura y dominio alcanzado en el país en lo que se refiere al desarrollo de la informática, se aprecia que es viable y factible la creación de soluciones propias para contribuir al progreso de las tecnologías CAD para la industria nacional, lo que se verifica en la práctica por resultados parciales de proyecto que se han obtenido; el primero de estos fue la aplicación ASIXMEC con funcionalidades para el modelado paramétrico y visualización de piezas en 3D (Peña Peñate 2015); posteriormente, como parte de los trabajos desarrollados por el grupo de investigación SIPII, se implementaron soluciones informáticas para el diseño asistido por computadora asociadas a los siguientes trabajos de diploma:

- *"Componente para el modelado de engranajes cilíndricos de dientes rectos"* del Ingeniero en Ciencias Informáticas Sandy García Santos en la cual se presentan los resultados del proceso de desarrollo de un componente para generar el modelo geométrico de engranajes cilíndricos con dientes rectos (Santos 2015).
- *"Componente para el modelado de árboles escalonados para el Sistema CAD 2D"* desarrollado por el Ingeniero Abel Fernández Ferrer, que automatiza el proceso de modelado de este tipo de componente mecánico (Fernández Ferrer 2015).
- *"Componente para el modelado de vigas para el Sistema CAD 2D"* desarrollado por el Ingeniero Julio César Pérez González, como resultado de la cual se obtuvo un módulo para automatizar el modelado de vigas con perfiles diversos y la posibilidad de los cálculos para determinar las fuerzas cortantes y los momentos flectores ante diferentes estados de carga (González, Julio César Pérez 2016).
- "*Componente para acelerar el diseño de resortes helicoidales*" donde el ingeniero Gustavo García González obtiene con las herramientas disponibles de código abierto, un

componente para automatizar el diseño de resortes helicoidales que funcionen a compresión, tracción y torsión (González, Gustavo García 2016).

Los componentes obtenidos precisan integración, para su empleo como parte de un sistema de Diseño Asistido por Computadoras; aunque pueden explotarse de forma independiente utilizando un visor simplificado, es necesario que puedan ser acoplados en una plataforma, que garantice la comunicación entre los módulos y facilite la interacción de los usuarios con los mismos.

Un trabajo precedente titulado “Aplicación para visualizar entidades geométricas, topologías y modelos de componentes mecánicos (CAD2D)”, consistió en desarrollar una aplicación con funcionalidades para visualizar, exportar e importar datos con formatos STEP[3], BRep[4] e IGES[5] (Amador 2015); este resultado, para las condiciones de crecimiento de la aplicación, aún es insuficiente, pues no integra las funcionalidades básicas necesarias de un sistema CAD, como los mecanismos para asegurar la persistencia de datos durante una o varias sesiones de trabajo, la manipulación y edición de objetos o el dibujo paramétrico conectado al diseño basado en el historial de rasgos de la pieza *(History Base Feature)* (Spitz y Rappoport 2004).

La situación descrita motivó la realización de una investigación preliminar, en la que se estudiaron las funcionalidades de los sistemas comerciales Autodesk Inventor, SolidEdge, SolidWorks, CATIA y las arquitecturas de los desarrollos de código abierto FreeCAD y OCAF, para precisar las características y funcionalidades más importantes de los sistemas CAD; por los análisis realizados se pudo constatar que:

- Existen características y funcionalidades en los sistemas CAD modernos que son imprescindibles, como las que aseguran la manipulación y edición de objetos geométricos, la parametrización, persistencia de datos y la visualización.
- Las características señaladas, por tener un ámbito global para su empleo por otros módulos, debían encontrarse en el núcleo de la aplicación.
- Las características y funcionalidades mencionadas están implementadas en los códigos fuente de la aplicación FreeCAD, que para el modelado emplea las funciones de la tecnología *Open CASCADE*.

3 STEP es un formato de archivo CAD, utilizado para compartir modelos 3D entre usuarios con diferentes sistemas CAD (PCB-3D 2017).

4 El formato BREP se utiliza para almacenar un modelo 3D, ubicación de espacio y orientación (OPEN CASCADE S.A.S 2013).

5 (Initial Graphics Exchange Specification) IGES fue desarrollado para permitir el intercambio de datos (Wilson 1987).

Teniendo en cuenta los argumentos precedentes se formula el siguiente **problema de investigación:** ¿Cómo agrupar las funcionalidades básicas y generales, de aseguramiento al modelado paramétrico en dos y tres dimensiones, para su empleo como núcleo en sistemas de Diseño Asistido por Computadoras?

Para enfrentar los desafíos relacionados con el problema planteado se determinó, en el grupo de investigación, seguir a lo largo de todo el proceso de desarrollo los criterios metodológicos siguientes: transitar desde los problemas más simples a los más complejos y reutilizar concepciones y funcionalidades existentes en tecnologías de código abierto disponibles.

El **objeto de estudio** se centra en los núcleos de aplicaciones para el Diseño Asistido por Computadoras con aplicaciones industriales y se define como **campo de acción,** las funcionalidades básicas de carácter general requeridas por esos núcleos.

Para el desarrollo del trabajo se plantea como **objetivo general**: desarrollar un núcleo para aplicaciones de Diseño Asistido por Computadoras, con las funcionalidades generales y básicas que aseguren el modelado paramétrico en dos y tres dimensiones, basándose en los sistemas y tecnologías de código abierto disponibles (FreeCAD, Coin3D, Qt y Open CASCADE Community Edition).

Del objetivo general se derivan los siguientes **objetivos específicos**:

1) Identificar las características esenciales y las funcionalidades básicas de carácter general, necesarias para conformar un núcleo de aplicaciones CAD.
2) Realizar el diseño conceptual del núcleo considerando su interacción con módulos e interfaces previstas.
3) Implementar el núcleo.
4) Realizar pruebas al núcleo.

Para dar cumplimiento a los objetivos específicos se proponen las siguientes **tareas de investigación:**

- Asimilación de los conceptos y tecnologías requeridos para el desarrollo del núcleo (Qt, Open CASCADE, Coin3D, SoQt).
- Definición del perfil y diseño de la investigación.
- Búsqueda y revisión bibliográfica sobre el objeto de investigación con énfasis en los temas de Ingeniería de Software que se requieran como los diseños arquitectónicos.

- Fundamentación de las necesidades y requerimientos de proyecto que avalan la conformación del núcleo.
- Análisis de los códigos fuentes disponibles para inferir información de sus arquitecturas.
- Estudio de las funcionalidades existentes en aplicaciones para el diseño asistido por computadoras comerciales, que permitan inferir información sobre la arquitectura de los mismos.
- Estimación de las funcionalidades requeridas, para garantizar un funcionamiento eficiente de las aplicaciones CAD en que sea empleado el núcleo.
- Proyección de las características esenciales del núcleo[6].
- Definición de la arquitectura del núcleo (definición de las funcionalidades para la serialización, exportación e importación, visualización, edición, selección y conectividad con los módulos) lo que implica además definir los patrones de diseño con los que cumplirá.
- Estudio de los aspectos de la metodología de desarrollo de software (AUP-UCI) que se aplicarán en el proceso de desarrollo.
- Determinación de la estructura de clases del núcleo.
- Modelado del flujo de datos del núcleo.
- Diseño de la interfaz gráfica del núcleo.
- Implementación del núcleo (incluye las funcionalidades para gestionar e integrar los módulos).
- Realización de pruebas al núcleo (de las diferentes funcionalidades, de integración y gestión de módulos y de compatibilidad para aplicaciones de diseño 2D y 3D).

Como resultados a obtener se proyectó:

1. Un núcleo para aplicaciones de Diseño Asistido por Computadoras, con funcionalidades generales y básicas para asegurar el modelado paramétrico en dos y tres dimensiones.
2. Documento contentivo de la información, sobre el proceso de trabajo y los resultados obtenidos.

6 Las funcionalidades básicas y las características del núcleo representan los criterios principales para el diseño del mismo.

Para resolver y dar cumplimiento a los objetivos y las tareas propuestas se emplearon los siguientes métodos:

Análisis y síntesis: Para determinar el estado del arte relacionado con el tema y conformar la fundamentación teórica del trabajo, para extraer información de los código fuente de la aplicación FreeCAD y de las funcionalidades de sistemas comerciales para el Diseño Asistido por Computadoras.

Modelado: Durante la conformación de los diagramas y la generación de los artefactos.

Observación: Durante el estudio de las funcionalidades y captura de requisitos a sistemas comerciales y de código abierto para el Diseño Asistido por Computadoras como SolidEdge, AutoCAD, CATIA, Inventor y FreeCAD.

Experimentación: Durante la realización de pruebas al núcleo.

Para el cumplimiento de las pautas establecidas en el diseño de investigación, conformado con la información de las pesquisas preliminares fue necesario procesar nueva información; en el apartado que sigue se exponen algunos de los aspectos generales sobre la composición de los sistemas para el Diseño Asistido por Computadoras.

1.2 Aspectos generales sobre la composición de las aplicaciones para el Diseño Asistido por Computadoras

Para resolver cualquier problema relacionado con las tecnologías para el Diseño Asistido por Computadoras, es necesario comprender la estructura de los sistemas que las conforman y como funcionan, así como los algoritmos y métodos que tienen implementados; mediante la observación de la forma en que estos sistemas funcionan, es posible obtener información importante mediante la inferencia y la estimación, por lo que empezaremos a relacionar los aspectos que como regla, aparecen en ellos.

Lo primero que se puede apreciar es que existe una gran diversidad de sistemas para el diseño asistido por computadoras con aplicaciones industriales y que a lo largo de más de tres décadas han estado en un proceso de desarrollo, en el que se han multiplicado sus funcionalidades y niveles de complejidad en muchos sentidos; una rama importante que parece marcar la ruta de avance son los denominados "Mechanical Computer Aided Design" (MCAD), entre los que se pueden mencionar las aplicaciones AutoCAD e Inventor de Autodesk, Inc.; SolidWorks de SolidWorks Corp., y que en la actualidad es una filial de Dassault Systèmes S.A; CATIA de Dassault Systèmes S.A; Solid Edge de Siemens PLM Software, entre otros.

En los MCAD se distingue como aspecto importante la modularidad, generalmente con la presencia de tres módulos que son Pieza, Ensamble y Dibujo; aunque algunos incorporan otros módulos o agrupan funcionalidades en cajas de herramientas ("toolboxes") estos son los principales en los que se basa el diseño mecánico.

Para poder asegurar las operaciones necesarias en el proceso de diseño, se requiere en estas aplicaciones una estructura sistémica con subsistemas asociados, por ejemplo, para elaborar las geometrías en dos dimensiones y los modelos topológicos en tres dimensiones es imprescindible un subsistema que encapsule algoritmos para el modelado; para que los modelos se vean en la pantalla del ordenador se precisa de un subsistema que agrupe las funcionalidades de visualización; del mismo modo debe ocurrir para manipular y operar con los datos, tanto para que persistan en un formato nativo, como para su intercambio con otros sistemas; más información sobre estos aspectos se puede precisar en la documentación de Open CASCADE Technology.

Sobre criterios mencionados, en el apartado sobre los componentes CAD/CAM[7] de la referencia (Ligero 2008), se relacionan una serie de aspectos a los que se denomina componentes, en referencia a los conjuntos de funcionalidades que están presentes en los sistemas CAD/CAM; de estos pertenecen a los sistemas de Diseño Asistido por Computadoras los siguientes:

- **Modelado geométrico:** Agrupa las funcionalidades basadas en los métodos para construir entidades geométricas y topologías.
- **Técnicas de visualización:** Destinadas a generar las imágenes de los modelos construidos, se puede apreciar que los sistemas CAD tienen implementados varios modos para la visualización, por ejemplo, "wireframe" (alámbrico) y "shade" (sombreado).
- **Técnicas de interacción gráfica:** Constituye el soporte a la entrada de datos para el modelado y agrupa las técnicas de posicionamiento y selección; las técnicas de posicionamiento se utilizan para la introducción de coordenadas en el plano y el espacio. Las técnicas de selección permiten la identificación interactiva de los componentes topológicos del modelo (vertex, edge, wire, face, solid) por lo que son esenciales durante la edición.
- **Interfaz de usuario:** Conjunto de subsistemas gráficos para hacer viable la interacción del usuario con el sistema, la eficiencia en el uso de la herramienta depende en gran medida del diseño de este componente.
- **Persistencia de datos:** Para almacenar la información del modelo y datos del diseño.

7 Computer Aided Manufacturing

Existen diversos puntos de vista sobre la estructura de los sistemas para el Diseño Asistido por Computadoras, la anteriormente referenciada expone una de ellas; en el siguiente apartado se profundiza en las características y funcionalidades que están presentes en los mismos.

1.2.1 Apreciación de las funcionalidades generales existentes en los sistemas para el Diseño Asistido por Computadoras

Veamos algunos de los criterios que sobre las funcionalidades generales de los sistemas para el Diseño Asistido por Computadoras ofrecen algunas de las fuentes consultadas; en la referencia (Poblet 1986) se cuentan las siguientes:

- Definición interactiva del objeto
- Visualización múltiple
- Cálculo de propiedades
- Modificación del modelo
- Generación de planos y documentación
- Conexión con sistemas de Fabricación Asistida por Computadoras.

Se les subdivide en herramientas de dibujo en dos dimensiones y modeladores en tres dimensiones; las herramientas de dibujo en dos dimensiones permiten definir y manipular elementos de geometría en el plano, lo que se puede realizar a través de una interfaz gráfica; los objetos que manipulan son entidades geométricas vectoriales como puntos, líneas, arcos, polígonos, para lo que se requiere el empleo de coordenadas en el plano (dos dimensiones).

Se reporta que algunos sistemas para el dibujo en dos dimensiones, que emplean diferentes niveles o capas para conseguir efectos tridimensionales, referenciados como de dos dimensiones y media (2+½D) son capaces de trabajar con dibujos planos definidos con coordenadas en el plano, pero manejan elementos del modelo en diferentes elevaciones (Ligero 2008).

Los programas que trabajan con dibujos espaciales, sus puntos están definidos por tres coordenadas y logran representaciones en tres dimensiones; los más complejos añaden superficies y sólidos (Ligero 2008; Poblet 1986).

En el caso de (Torres 2005) se esboza una estructura con los siguientes componentes:

- **Representación del Modelo:** es la representación computacional del objeto que se está diseñando, contiene toda la información necesaria para describir el objeto, tanto a nivel geométrico, como de características; es el elemento central del sistema, el resto de los

componentes trabajan sobre él. Por tanto determinará las propiedades y limitaciones del sistema CAD.

- **Subsistema de edición:** permite la creación y edición del modelo, bien a nivel geométrico o especificando propiedades abstractas del sistema. En cualquier caso la edición debe ser interactiva para facilitar la exploración de posibilidades.
- **Subsistema de visualización:** se encarga de generar imágenes del modelo. Normalmente interesa poder realizar distintas representaciones, contiene más de un modo de representar gráficamente el ente que se está diseñando y permite visualizaciones rápidas durante la edición.
- **Subsistema de cálculo:** proporciona el cálculo de propiedades del modelo y la realización de simulaciones.
- **Subsistema de documentación:** se encarga de generar la documentación del modelo.

Por su parte (Rojas 2014) plantea que las características generales de los sistemas CAD/CAE:

- Capacidad de generar soluciones óptimas según los tipos de aplicación.
- Desarrollo de prototipos virtuales que constituyen alternativas de los físicos.
- Ingeniería concurrente *ON-LINE* (trabajo multidisciplinario vía red, con niveles de acceso y con geoprocesamiento referenciado).
- Arquitectura abierta del software (posibilidad de personalizar y generar programas complementarios "*glue functions*").
- Ingeniería inversa (obtener un modelo CAD a partir del escaneado tridimensional de una pieza real).
- Intercambio estandarizado de formatos de archivos para el trabajo multiplataforma (*run anywhere*).
- Pantallas de trabajo (*workspace*) compartidas con diferentes aplicaciones y programas adicionales (*plug-ins*).

De modo particular, en lo que respecta a los sistemas MCAD se pueden distinguir funcionalidades con carácter global y local, las primeras se les puede observar en cualquiera de los módulos que se activen, mientras que las segundas solamente se pueden ubicar en algunos de ellos.

Se aprecia que una estructura lógica para implementar las funcionalidades de ámbito global en las aplicaciones CAD, es edificar el sistema con un núcleo que las contenga; esa parece ser la

práctica seguida en los sistemas comerciales, pero ¿cuáles de estas pueden estar implementadas en el núcleo? evidentemente, las de modelado son de ámbito local y se puede considerar que deben estar implementadas en los módulos dedicados a construir los modelos, es decir, en los de Pieza y en el de Ensamble, también en los que constituyen "aceleradores de diseño", que en aplicaciones como Inventor y Solid Edge se ubican dentro del módulo de Ensamble.

El subsistema que agrupa los procedimientos de visualización es de los que tiene carácter global, en este caso se cuentan las funcionalidades de resaltado, selección y manipulación de las entidades geométricas y topologías; el subsistema que contiene las funcionalidades para garantizar la manipulación de los datos también es global, pues es utilizado desde cualquier módulo; de modo que estas dos estructuras pueden encontrarse implementadas en un núcleo central.

En el caso del subsistema de visualización la funcionalidad de resaltado permite al usuario distinguir el elemento topológico que señala el cursor del "Mouse", lo que es imprescindible para seleccionar los mencionados elementos y garantizar otras operaciones como las de edición; por manipulación se entiende la traslación y la rotación, lo cual está asociado también a la posibilidad de resaltar y seleccionar.

Para garantizar operaciones como la traslación y la rotación, es imprescindible la existencia de un mecanismo para la gestión de las propiedades asociadas a los objetos topológicos y geométricos, el cual también tiene carácter global, por lo que debe estar implementado también en el núcleo; este es uno de los que garantiza la parametrización en los sistemas CAD modernos; la otra parte de la parametrización se asegura con lo que en la literatura especializada se denomina "Geometric Constraints Solver" que son "sistemas inteligentes" destinados a encontrar las condiciones de satisfacción entre elementos geométricos y las restricciones impuestas a estos.

Durante el análisis del código fuente de la aplicación FreeCAD se conoció, que en ese caso concreto, las funcionalidades del subsistema destinado a la manipulación de los datos está compuesto por el mecanismo de gestión de propiedades de los objetos, que garantiza la persistencia, la posibilidad de actualizar los modelos geométricos después de alguna modificación y en alguna medida las operaciones Deshacer y Rehacer ("Redo" y "Undo"), los que representan beneficios del paradigma de parametrización.

Se puede asumir, de acuerdo a las observaciones realizadas y las fuentes consultadas, **que las funcionalidades generales de los sistemas CAD pueden estar concentradas en un núcleo, pues son utilizadas por todos los módulos que las integran; otras funcionalidades específicas deben estar implementadas en los módulos garantizando las premisas de los sistemas estructurados**. La práctica de construir las aplicaciones teniendo como centro un

núcleo que contenga las funciones más generales, parece ser también el criterio lógico seguido en los sistemas de carácter comercial.

Con el nivel de información acumulado durante los análisis precedentes, se hace necesario abordar el tema desde el punto de vista de la Ingeniería de Software, pues el propósito previsto de desarrollar un núcleo para aplicaciones CAD, requiere de una concepción que garantice un desarrollo robusto pero flexible, y para eso es necesario profundizar en los fundamentos de los diseños arquitectónicos.

1.3 Acerca de los diseños arquitectónicos

Según Ian Sommerville en (Sommerville 2011), “...El diseño arquitectónico es un proceso creativo en el cual se diseña una organización del sistema que cubrirá los requerimientos funcionales y no funcionales de este...”; de acuerdo a lo planteado es durante el proceso de diseño arquitectónico que se toman las decisiones fundamentales que definirán al sistema y su proceso de desarrollo.

Aun cuando cada sistema posee una estructura única, los que se encuentran en el mismo dominio de aplicación están compuestos por arquitecturas similares, lo que permite apoyar el diseño del sistema en uno o varios patrones arquitectónicos (Sommerville 2011).

En la fuente (Pressman 2003) "...un patrón describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto especifico..."; en el presente trabajo se acoge la definición de Sommerville en (software-engineering-book 2015) en la que plantea que un patrón arquitectónico es una descripción de las buenas prácticas de diseño, que han sido probadas en diferentes entornos.

A continuación se expone una breve descripción de los principales patrones arquitectónicos con sus ventajas y desventajas.

Repositorio

Todos los datos de un sistema se administran en un repositorio central accesible a todos los componentes del sistema.

Ventajas

Cuando se comparten grandes cantidades de datos, el modelo de repositorio de uso compartido se utiliza con mayor frecuencia, esto es un mecanismo eficiente de intercambio de datos (software-engineering-book 2015).

Desventajas

Los componentes no interactúan directamente, solo a través del repositorio. El repositorio es un único punto de fallo, por lo que los problemas en el repositorio afectan a todo el sistema (software-engineering-book 2015).

Se debe utilizar este patrón cuando se tenga un sistema en el que se generen grandes volúmenes de información que deban almacenarse durante mucho tiempo. También puede utilizarse en sistemas basados en datos en los que la inclusión de datos en el repositorio activa una acción o herramienta (software-engineering-book 2015).

Modelo-Vista-Controlador (MVC)

El sistema está compuesto por tres componentes (Modelo, Vista, Controlador) que interactúan entre sí; el modelo gestiona los datos y las operaciones asociadas a estos, la vista define y administra cómo se muestran los datos al usuario y el controlador gestiona la interacción del usuario (por ejemplo, pulsaciones de teclas, clics del ratón) y la envía a la vista y al modelo.

Ventajas

Permite que los datos cambien independientemente de su representación y viceversa. Apoya la presentación de los mismos datos de diferentes maneras con los cambios realizados en una representación mostrada en todos ellos (software-engineering-book 2015).

Desventajas

Puede implicar código adicional y complejidad de código cuando el modelo de datos y las interacciones son simples.

Se utiliza cuando hay varias maneras de ver e interactuar con los datos. También se utiliza cuando se desconocen los futuros requisitos para la interacción y la presentación de los datos (software-engineering-book 2015).

Arquitectura en Capas

En la arquitectura en capas cada una de estas ejecuta operaciones que se aproximan progresivamente al conjunto de instrucciones de la máquina. En la capa externa, los componentes atienden las operaciones de la interfaz de usuario. En la interna, los componentes realizan la interfaz con el sistema operativo. Las capas intermedias proveen servicios útiles y funciones de software de aplicación.

Ventajas

Este sistema basado en capas nos permite niveles de abstracción crecientes, lo que facilita la división de problemas complejos en una secuencia de pasos incrementales. A medida que se desarrolla una capa, algunos de los servicios proporcionados por esa capa pueden estar disponibles para los usuarios.

Desventajas

La desventaja de la aproximación por capas es que la estructuración de los sistemas puede resultar difícil; las capas internas pueden proporcionar facilidades básicas, tales como gestión de ficheros, que son requeridas por todos los niveles, pero los servicios requeridos por un usuario del nivel superior tendrán que pasar por las capas adyacentes para tener acceso a los servicios proporcionados por los niveles inferiores (Sommerville 2005).

Por lo referenciado en el presente apartado, se puede concluir que la concepción arquitectónica de una aplicación depende de los requerimientos relacionados con el problema a resolver; si además se toman en cuenta los elementos que componen el perfil de la investigación y los resultados de los análisis obtenidos en el presente apartado, es previsible que el diseño más adecuado a las condiciones planteadas esté basado fundamentalmente en una Arquitectura por capas.

Otros aspectos a considerar cuando se define la arquitectura de un sistema son los relacionados con el manejo de los datos, el marco de trabajo OCAF tiene definidas características sobre este particular, que se han estudiado para comprender los mecanismos y compararlos con los de la aplicación FreeCAD.

1.4 Estructuras de datos en los sistemas para el Diseño Asistido por Computadora

La forma en que se estructuran los datos en las aplicaciones CAD debe garantizar la persistencia de los mismos y asegurar otras funcionalidades importantes como la gestión de documentos, la organización de los datos y las operaciones de Rehacer y Deshacer; como se mencionó en 1.1, en software libre solamente se encuentra disponible el marco de trabajo OCAF, como parte componente de la tecnología Open CASCADE, el cual está concebido para desarrollar de forma rápida aplicaciones de diseño sofisticadas.

En la documentación de la fuente («Open CASCADE Technology: OCAF» 2016) se plantea que el desarrollo de una aplicación para el diseño asistido por computadora, requiere tomar en consideración los siguientes aspectos estructurales:

- Diseño de la arquitectura de la aplicación (definición de los componentes de software y la forma en que cooperan).
- Definir el modelo de datos capaz de soportar la funcionalidad requerida (una aplicación de diseño opera sobre datos mantenidos durante toda la sesión de trabajo del usuario final).
- Estructura del software para:
    - Sincronizar la pantalla con los datos (los comandos que modifican los objetos deben actualizar las vistas).
    - Soportar comandos deshacer-rehacer (esta característica debe tenerse en cuenta en los primeros pasos del proceso de diseño).
- Implementar la función para guardar los datos (si la aplicación tiene un ciclo de vida largo, la compatibilidad de los datos entre las versiones de la aplicación tiene que ser abordada).
- Construir la interfaz de usuario de la aplicación.

En el apartado que sigue se describen los aspectos esenciales de OCAF.

1.4.1 Aspectos generales del marco de trabajo "Open CASCADE Application Framework" *(OCAF)*

En la revisión bibliográfica realizada se encuentra la definición de "*Open CASCADE Application Framework*" (OCAF) como un conjunto de bibliotecas de clases escritas en el lenguaje de programación C++ («Open CASCADE Technology: OCAF» 2016), esto permite que aspectos complejos de las aplicaciones CAD puedan implementarse de manera relativamente fácil con las funciones que provee el marco de trabajo, la Tabla 1 contiene un cuadro comparativo de las implicaciones que tiene trabajar con OCAF o prescindiendo de él.

Tabla 1: Comparación entre el diseño de una aplicación CAD empleando o no OCAF.

Tareas de desarrollo	Comentarios	Sin OCAF	Con OCAF
Creación de geometría	Llamadas a las bibliotecas de modelado	Creado por el usuario	Creado por el usuario
Organización de datos	Incluyen atributos específicos y proceso de modelado	Creado por el usuario	Simplificado
Guardar datos en un archivo	Noción del documento	Creado por el usuario	Previsto
Gestión de documentos		Creado por el usuario	Previsto
Infraestructura de aplicaciones	Menú: nuevo archivo, abrir, cerrar, guardar y guardar como	Creado por el usuario	Previsto
Deshacer rehacer	Robusto, multinivel	Creado por el usuario	Previsto
Cuadros de diálogo específicos de la aplicación		Creado por el usuario	Creado por el usuario

La columna que indica "Sin OCAF" en la tabla anterior, ilustra la situación existente con la aplicación FreeCAD, en la que las tareas de desarrollo quedan bajo la responsabilidad del usuario para su creación, lo que constituye una desventaja importante; pero por otra parte, en un principio no existía dominio suficiente que asegurara la implementación de las funcionalidades de este marco de trabajo.

La situación existente en la aplicación FreeCAD fue mencionada por Werner Mayer, uno de los desarrolladores principales de FreeCAD, en un ***post del foro oficial de la aplicación el día 28 de abril de 2010***, en el que menciona las dificultades que enfrentaron principalmente con la tecnología de visualización de Open CASCADE y el marco de trabajo OCAF (Mayer 2010; FreeCAD 2017).

Esta realidad es la que motivó aferrarse a la idea de desarrollar una solución propia basada en la concepción estructural y funcional de FreeCAD; **precisamente era el código fuente de esta aplicación, el que se presentaba como más apropiado para servir de base en el trabajo de desarrollo**.

La descripción de OCAF que sigue ha servido para el estudio teórico, durante el proceso de investigación y desarrollo.

Descripción general de la arquitectura

El marco de trabajo OCAF proporciona un modelo arquitectónico Aplicación-Documento-Atributo *(fig.1)* orientado a objetos.

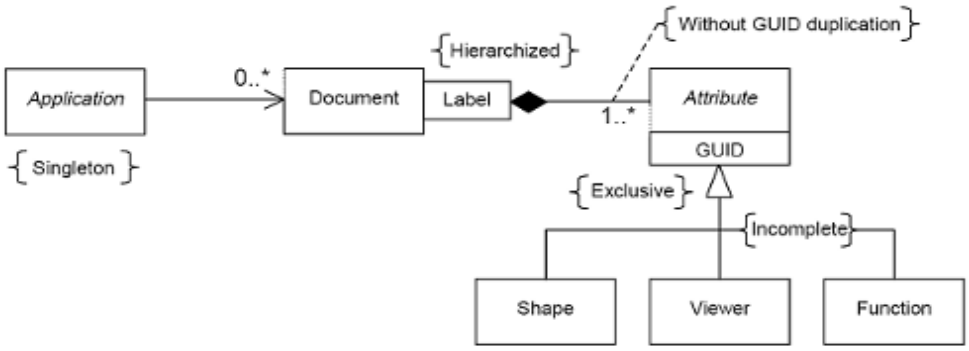

Fig 1: Modelo Aplicación-Documento-Atributo

Tomado de: www.Open CASCADE.com

La clase principal (*Application)* es abstracta y está encargada de manejar los documentos durante la sesión de trabajo e incluye los siguientes servicios:

- Creación de nuevos documentos
- Guardar y abrir los documentos
- Inicialización de vistas de los documentos

Una aplicación puede contar con varios documentos, implementados por la clase concreta *Document,* que es el contenedor de los datos de la aplicación. Los documentos ofrecen acceso al marco de datos y cumplen los siguientes propósitos:

- Administrar la notificación de cambios
- Actualizar enlaces externos
- Gestionar la persistencia y restauración de datos
- Guardar los nombres de las extensiones de software
- Administrar transacciones de comandos
- Administrar opciones deshacer y rehacer

Cada documento se guarda en un único archivo *ASCII* definido por su formato y extensión (OCAF proporciona un formato listo para usar).

Los datos de la aplicación se describen por atributos**,** que son instancias derivadas de la clase abstracta *Attribute,* organizados según el marco de datos OCAF. Este marco hace referencia a las agregaciones de atributos utilizando identificadores persistentes en una misma jerarquía. Una amplia gama de ellos son incluidos con OCAF, ejemplos:

- Atributos estándar (*Standard attributes*) permiten operar con datos simples en el marco de datos (ejemplo: entero, real, cadena, matriz de datos), realizar funciones auxiliares, crear dependencias (ejemplo: referencia, nodo de árbol).
- Atributos de forma (*Shape attributes*) contienen la geometría de todo el modelo o sus elementos, incluida la referencia a las formas y el seguimiento de su evolución.
- Atributos geométricos tales como:
  - Sistemas de referencia: puntos, ejes y plano.
  - Restricciones: tangente a, distancia, ángulo, etcétera.
- Atributos de usuario (*User attributes*) es decir, atributos tipos por la aplicación.
- Atributos de visualización (*Visualization attributes*) permiten la visualización de la información almacenada en el marco de datos, la representación visual de los objetos y otra información visual auxiliar, que se necesite para la representación de datos gráficos.
- Servicios de función (*Function services*) el propósito de estos atributos es la reconstrucción de objetos después que son modificados (parametrización de los modelos). Mientras el documento administra la notificación de cambios, una función gestiona la propagación de estos cambios. El mecanismo de función proporciona enlaces entre funciones y llamadas a varios algoritmos.

Con la información antes expuesta se resumen los aspectos principales de OCAF, el cual se basa en un modelo uniforme de llave de referencia que proporcionan una identificación persistente de los datos, que incluyen la geometría y se implementan como atributos adjuntos a las claves de referencia. El nombre topológico mantiene la geometría seleccionada adjunta a las llaves de referencia en los modelos parametrizados. En muchas aplicaciones, el formato de datos proporcionado con OCAF no necesita ser extendido. OCAF no está acoplado a las bibliotecas de modelado subyacentes.

El marco de datos

El marco de datos de OCAF es un mecanismo de la tecnología *Open CASCADE* basado en el ***modelo llave de referencia*** para una estructura de árbol, que ofrece un entorno único en el que se pueden manejar los datos de diferentes componentes de la aplicación, lo que permite intercambiarlos y modificarlos de forma sencilla, consistente, con un nivel máximo de información y semántica estable.

Los elementos básicos de este enfoque son:

- La marca, índice o "tag"
- La etiqueta o "label"
- El atributo o "attribute"

Como se ha mencionado anteriormente, la primera etiqueta en un marco es la etiqueta raíz del árbol. Cada etiqueta cuenta con un índice expresado como un valor entero. Una etiqueta se define de forma única por una entrada expresada como una lista de índices desde la raíz, por ejemplo 0: 1: 2: 1.

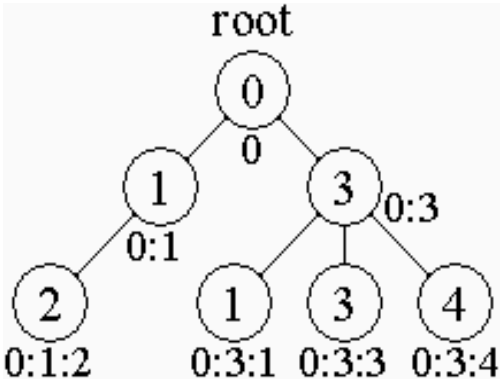

Fig 2: Un modelo de marco simple

Tomado de: www.Open CASCADE.com

Los círculos contienen el índice de las etiquetas correspondientes *(fig.2)*. Las listas de índices se encuentran debajo de los círculos. La etiqueta raíz siempre tiene un índice cero.

Etiqueta

Una etiqueta es un entero, que identifica una etiqueta de dos maneras:

- Identificación relativa
- Identificación absoluta.

En la identificación relativa, el índice de una etiqueta tiene un significado relativo a la etiqueta del padre solamente. Para una etiqueta específica, puede tener, por ejemplo, cuatro etiquetas hijas identificadas por los índices 2, 7, 18, 100. Al utilizar la identificación relativa, debemos asegurarnos de tener un ámbito seguro para establecer atributos.

En la identificación absoluta, el lugar de una etiqueta en el marco de datos se especifica sin ambigüedad por una lista de índices separados por dos puntos de todas las etiquetas en cuestión a la raíz del marco de datos. Esta lista permite la recuperación de la entrada correspondiente a una etiqueta específica.

Ejemplo de una estructura de datos

Con el objetivo de lograr una mayor comprensión del funcionamiento de OCAF, a continuación se expone un ejemplo de una estructura de datos para el diseño de lámparas de mesa *(fig.3)*, tomado de la referencia (OCAF 2016).

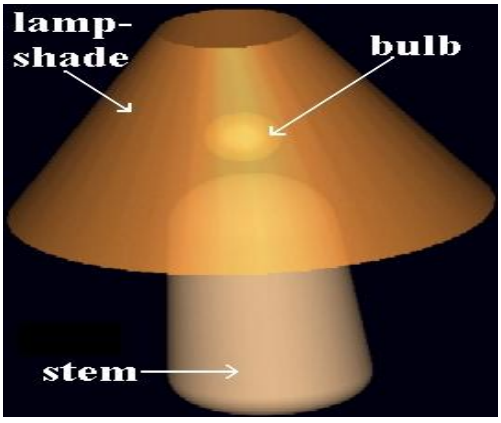

Fig 3: Lámpara de Mesa

Tomado de: www.opencascade.com

La etiqueta raíz no puede tener etiquetas hermanas. Por consiguiente, varias lámparas en el marco de datos corresponden a las subetiquetas de la etiqueta raíz. Esto permite evitar cualquier confusión entre las lámparas de mesa en el marco de datos. Las diferentes partes de la lámpara tienen diferentes materiales, colores y otros atributos, por lo que se asigna una etiqueta secundaria de la lámpara con los índices específicos para cada subunidad de la lámpara:

- Una etiqueta para la forma de la lámpara *("lamp-shape")* con índice 1.
- Una etiqueta de la bombilla *("bulb")* con índice 2.
- Una etiqueta de tallo *("stem")* con índice 3.

Los índices de las etiquetas se eligen a voluntad, son solo identificadores de las partes de la lámpara. Ahora puede refinar todas las unidades: ajustando geometría, color, material y otra información sobre la lámpara o sus partes a la etiqueta especificada. Esta información se coloca en atributos especiales dentro de la etiqueta.

Por lo tanto, después de que el usuario cambia el diseño de la lámpara, solo se cambian los atributos correspondientes, pero se mantiene la estructura de la etiqueta. La forma de la lámpara debe ser recreada por los nuevos valores de sus atributos y los atributos de la forma de la lámpara deben referirse a una nueva forma.

Cada hijo de la etiqueta raíz contiene un atributo de forma de lámpara y hace referencia a las subetiquetas que contienen información de diseño sobre subunidades correspondientes *(fig.4)*.

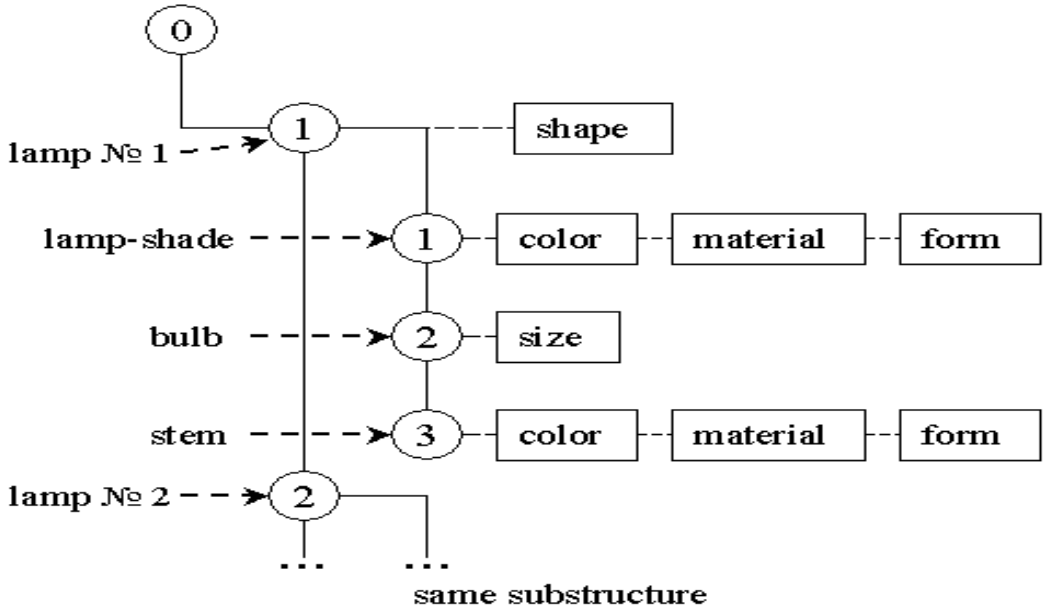

Fig 4: Estructura de un documento de las lámparas de mesa

Tomado de: www.Open CASCADE.com

Mecanismo de transacción

El marco de datos también proporciona un mecanismo de transacción, inspirado en los sistemas de gestión de bases de datos: los datos se modifican dentro de una transacción que es terminada por un "*Commit*" si las modificaciones son validadas o por un "*Abort*" si las modificaciones son abandonadas y regresa al estado que estaba antes de la transacción. Este mecanismo es extremadamente útil para:

- Asegurar las operaciones de edición (si se produce un error, la transacción se abandona y la estructura conserva su integridad)
- Simplificar la ejecución de la función de Cancelar (cuando el usuario final inicia un comando, la aplicación puede iniciar una transacción y operar directamente en la estructura de datos; abandonar la acción hace que la transacción sea abortada)
- Ejecutar Deshacer (durante la confirmación, las modificaciones se registran con el fin de ser capaz de restaurar los datos a su estado anterior)

El mecanismo de transacción simplemente administra una copia de seguridad de los atributos. Durante una transacción, los atributos se copian antes de su primera modificación; si la transacción es validada, la copia se destruye; si la transacción se abandona, el atributo se restablece a su valor inicial (cuando se añaden o eliminan atributos, la operación se invierte simplemente).

Las transacciones están centradas en el documento, es decir, la aplicación inicia una transacción en un documento. Por lo tanto, la modificación de la referencia de un documento y la actualización de uno de sus documentos de referencia requiere dos transacciones, incluso si ambas operaciones se realizan en la misma sesión de trabajo.

Además de los tratamientos descritos para la manipulación de datos, se requiere que estos puedan continuar existiendo en otras sesiones de trabajo, de lo contrario se perdería la información creada, por eso tienen especial importancia los mecanismos para asegurar la Persistencia y Serialización.

1.5 Persistencia y serialización de los datos

Durante la ejecución de un programa informático, la información procesada es almacenada de manera temporal en la memoria del ordenador; es por ello que es necesario contar con un mecanismo que realice la persistencia[8] de dicha información para su posterior uso. Durante la ejecución de este mecanismo, se suele involucrar un proceso de serialización de los datos a un archivo, una base de datos o a otro medio de almacenamiento. Estos mecanismos pueden ser creados por el desarrollador o proporcionados por un framework o biblioteca.

8 Se entiende por persistencia la acción de preservar la información de un objeto de forma permanente (guardar), a su vez también se refiere a poder recuperar la información del mismo (leer) para que pueda ser nuevamente utilizada.

En la actualidad es muy difundida la utilización de “Extensible Markup Language” (XML) como formato de persistencia para guardar las características de los elementos. Durante el análisis de la aplicación FreeCAD realizado en el epígrafe 1.2.1 se conocío que el mecanismo de gestión de propiedades garantiza la persistencia; en este caso especifico se realiza mediante el empleo de un archivo con formato XML.

Debido a su importancia a continuación se expone un breve análisis de las tecnologías para la visualización.

1.6 Visualización

La visualización se define por J. Foley y B. Ribarsky como “...visualization might be the binding (or mapping) of data to representations that can be perceived. The types of bindings could be visual, auditory, tactile, etc., or a combination of these...” (Ribarsky y Foley 1994), es decir, que la visualización no es otra cosa que el procedimiento mediante el que se representa la información contenida en los datos de un modelo determinado.

Según J. Drucker la visualización gráfica puede abarcar todo tipo de imagen, así como, ilustraciones y esquemas de datos representativos (Drucker 2011). La visualización de la información transforma datos abstractos y fenómenos complejos de la realidad en mensajes visibles, hace posible que los individuos observen datos y fenómenos que son directamente inaprensibles y así comprender la información que se encuentra oculta.

En el tratamiento de estas imágenes se encuentran diferentes formatos y tipos de gráficos. Los programas que manejan gráficos, básicamente lo hacen por dos sistemas (Ligero 2008):

- **Gráficos de mapa de bits (bit-map):** a través de una trama de puntos que contiene los valores (colores) de cada punto de la pantalla.
- **Gráficos vectoriales:** a través de tablas de coordenadas que definen los datos geométricos de cada objeto básico del dibujo.

En la referencia (Ligero 2008) se platea que “...Los programas CAD manejan gráficos vectoriales, que al estar definidos matemáticamente, se pueden editar sin perder exactitud y no dependen del equipo. La calidad del trazado en papel depende solo de la calidad del trazador de plumillas (plotter) o impresora que se utilice...”.

La visualización es un procedimiento de importancia fundamental en los sistemas de Diseño Asistido por Computadora pues ofrece una salida visual para representar los modelos geométricos de piezas, estructuras y conjuntos ensamblados que constituyen prototipos virtuales de objetos

físicos existentes o en proyección, incluso en forma de imágenes o animaciones fotorrealistas; algunas de las tecnologías para la visualización se describen en el siguiente apartado.

1.6.1 Tecnologías de Visualización

Con el avance de los sistemas de cómputo y el hardware para procesamiento de video, han surgido tecnologías para la visualización en dos y tres dimensiones algunas de las cuales son:

OpenGL

OpenGL (***Open G****raphics* ***L****ibrary*) es una especificación estándar que define una *Application Programming Interface* (API) multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Desde su introducción en 1992, ha traído miles de aplicaciones a una amplia variedad de plataformas informáticas. Incorpora un amplio conjunto de funciones de renderizado, mapeo de textura, efectos especiales y otras poderosas de visualización. Los desarrolladores pueden aprovechar la potencia de OpenGL en todas las plataformas de escritorio y estación de trabajo populares, garantizando un amplio despliegue de aplicaciones (OpenGL 2016).

OpenSceneGraph

OpenSceneGraph es un conjunto de herramientas de alto nivel para el desarrollo eficaz de gráficos 3D, usada por desarrolladores de aplicaciones en campos como la simulación visual, deportes, realidad virtual, modelado y visualización científica; está escrito enteramente en el estándar C++ y OpenGL, funciona con todas las plataformas de Windows, OSX, GNU/Linux, IRIX, Solaris, HP-Ux, AIX y sistemas operativos FreeBSD (OpenSceneGraph 2016).

VTK

VTK es un kit de herramientas de visualización, posee un sistema orientado a objetos para el procesamiento y la visualización de imágenes en tres dimensiones; está libremente disponible para gráficos 3D, modelado, procesamiento de imágenes, renderizado volumétrico, visualización científica y visualización de información; incluye varias capas de interfaz interpretadas, tales como Tcl/Tk, Java y Python; soporta una amplia variedad de algoritmos de visualización: escalar, vector, tensor, textura, y métodos volumétricos. También soporta técnicas de modelado avanzado como el modelado implícito, reducción de polígonos, suavizado de mallas, cortes y contorneo, entre otros. Por otra parte, docenas de algoritmos de imágenes son integrados en el sistema, lo cual permite la mezcla de datos y algoritmos gráficos de imágenes 2D/3D (VTK 2016).

Open Inventor y Coin3D

Open Inventor es una API de gráficos 2D y 3D orientada a objetos, destinada a compañías de software que quieran hacer uso de un *framework* 3D para construir y vender aplicaciones de alto rendimiento, está escrita en C++ y OpenGL provee una capa de alto nivel de programación. Por su parte, Coin3D es un software libre completamente compatible con la API de Open Inventor versión 2.1 y es publicada bajo la licencia BSD 3 («Coin3D» 2016).

La aplicación Salome-Meca integra visores basados en diferentes tecnologías de visualización como Open CASCADE, Vtk y OpenGL; FreeCAD realiza la visualización con Coin3D, cuya eficiencia se reporta en la literatura es superior a la que garantiza OCC (FreeCAD 2016b).

Uno de los criterios asumidos en el proceso de investigación y desarrollo en el que se inserta el presente trabajo, fue experimentar las diferentes tecnologías de visualización como se hace en Salome-Meca, pero empezando con Coin3D como se implementa en FreeCAD; en el momento de redactar el presente documento la tecnología empleada es Coin3D, pero una de las recomendaciones derivada, es implementar también visores de OCC, Vtk y OpenGL para evaluar los niveles de efectividad y de eficiencia en los desarrollos que se ejecutan.

En este punto se impone la revisión sobre los aspectos que pueden sustentar el proceder ordenado de la etapa de desarrollo que se previó, los que se exponen en el siguiente apartado.

1.7 Acerca de la metodología para el desarrollo

Proceso Unificado Ágil (*Agile Unified Process*, AUP) es un enfoque de modelado híbrido creado por Scott Ambler cuando combinó el *Rational Unified Process* (RUP) con los métodos ágiles. Mediante esta combinación Ambler creó un marco sólido de procesos que se puede aplicar a todo tipo de proyectos de software, grandes o pequeños (Sánchez 2015).

Los principios que sustentan AUP son (Sánchez 2015):

- La mayoría de la gente no va a leer documentación detallada. Sin embargo, se necesitará orientación y formación de vez en cuando.
- La descripción del proyecto debe ser en unas pocas páginas.
- Se ajusta a los valores y principios descritos en la Alianza Ágil.
- El proyecto debe centrarse en ofrecer valor esencial en lugar de características innecesarias.
- Los desarrolladores deben estar libres de utilizar las herramientas más adecuadas para la tarea en cuestión, en lugar de cumplir con un decreto.

En la Universidad de las Ciencias Informáticas (UCI) se realizó una variación a dicha metodología con el propósito de normalizar el proceso de desarrollo de software dentro de todos sus centros productivos (Sánchez 2015).

La variación AUP-UCI define sus cuatro fases como:

Tabla 2: Fases AUP-UCI

Fases AUP	Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración Construcción Transición	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y libera el producto. Durante esta se fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Para modelar el sistema en los proyectos productivos AUP-UCI define cuatro escenarios:

- Escenario No 1: Proyectos que modelen el negocio con Casos de Uso del Negocio solo pueden modelar el sistema con Casos de Uso del Sistema.
- Escenario No 2: Proyectos que modelen el negocio con Modelo Conceptual solo pueden modelar el sistema con Casos de Uso del Sistema.
- Escenario No 3: Proyectos que modelen el negocio con Diagrama de Procesos del Negocio solo pueden modelar el sistema con Diagrama de Requisitos por Proceso.
- Escenario No 4: Proyectos que no modelen negocio solo pueden modelar el sistema con Historias de Usuario.

La presente investigación posee un negocio muy bien definido y el cliente permanece siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos. Como guía para el desarrollo del trabajo de diploma será utilizada la metodología AUP en su variante UCI; basándose en los argumentos antes expuestos se seleccionó el cuarto escenario.

1.8 Tecnologías para el desarrollo. Framework y lenguajes

La evolución de las tecnologías informáticas ha llevado consigo un aumento exponencial de las herramientas y lenguajes que se emplean para el desarrollo de nuevos sistemas. Seguidamente se describen las características de las tecnologías empleadas en la presente investigación.

1.8.1 Lenguaje de programación C++

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C, abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos por lo que se considera un lenguaje híbrido multiparadigma. C++ es usado por millones de programadores en cada dominio de aplicación. Billones de líneas de C++ están actualmente desarrolladas. Muchos sistemas operativos poseen partes esenciales escritas en C++, como Windows, iOS, Linux, entre otros. Este lenguaje no fue diseñado con la computación numérica en mente, sin embargo, muchos programas de ingeniería, numéricos y científicos son realizados en C++. Este puede coexistir con código escrito en otro lenguaje. C++ es soportado por una variedad de librerías y conjuntos de herramientas, tales como Boost, Qt, Open CASCADE, Coind3D, OpenCV, entre otras (Stroustrup 2013).

1.8.2 Open CASCADE Technology

En el apartado 1.1 se expuso una breve descripción de **Open CASCADE Technology** un “Kernel” gráfico de código abierto destinado al desarrollo de sistemas CAD/CAE; *Open CASCADE Community Edition* (OCE) es la versión desarrollada por la comunidad, la cual es reconocida y aceptada por *OPEN CASCADE Company.* OCE es desarrollada a través de las recomendaciones de optimización de las versiones liberadas, mediante foros o en la propia página principal de desarrollo de esta tecnología («OCTT: Overview» 2017).

Debido a las potencialidades que ofrece OCE para el desarrollo de herramientas del tipo CAD/CAM/CAE y su actual empleo en el Sistema CAD 2D, sistema al que está destinado la presente investigación, se escoge OCE como *framework* de modelado.

1.8.3 Herramienta de Visualización

En la sección 1.6.1 del presente capitulo, se mencionaron algunas de las principales tecnologías para la visualización en dos y tres dimensiones; sobre la base de los argumentos antes expuestos se seleccionó Coin3D como herramienta inicial de visualización.

1.8.4 Framework de desarrollo

Qt es un *framework* de desarrollo de aplicaciones multiplataforma para escritorio, sistemas embebidos y dispositivos móviles. Compatible con Linux, OS X, Windows, VxWorks, QNX, Android, iOS, Blackberry, Sailfish OS y otros; contiene módulos para el desarrollo en áreas como redes, bases de datos, OpenGL, tecnologías web, sensores, protocolos de comunicación, procesamiento de XML y JSON, impresión, generación de PDF, entre otros («About Qt» 2016).

Algunas características que posee el *framework* Qt son:

- Constituye además una biblioteca para la creación de interfaces gráficas. Se distribuye bajo una licencia libre GPL, además de la LGPL, que permite su utilización gratuita con fines comerciales.
- Compatibilidad multiplataforma con un solo código fuente.
- Fácil de internacionalizar.
- Arquitectura lista para plugins.

Se escoge como *framework* de desarrollo debido a todas las ventajas antes mencionadas. Qt es utilizado en FreeCAD, lo que permite reutilizar parte del código de su núcleo, garantizando una mayor integración de sus módulos en la aplicación.

1.8.5 Sistema de control de versiones

Git es un sistema de control de versiones distribuido, libre y de código abierto, diseñado para manejar proyectos pequeños o muy grandes con rapidez y eficiencia (Torvalds y Hamano 2010).

La característica Git que realmente hace que se aparte de casi todos los otros SCM es su modelo de ramificación. Permite tener múltiples ramas locales que pueden ser totalmente independientes entre sí. La creación, la fusión y la supresión de esas líneas de desarrollo toma segundos.

Esto significa que se pueden ejecutar acciones como las indicadas en (Torvalds y Hamano 2010):

- **Conmutación de contexto sin fricción.** Crear una rama para probar una idea, realizar varios cambios, cambiar de nuevo a la rama de donde se ramificó, aplicar un parche, cambiar de nuevo a la rama donde se está experimentando y fíjarlo.
- **Reglas Basadas en el Rol.** Tener una rama que siempre contiene solo lo que va a la producción, otra que se fusiona en el trabajo para la prueba, y varias más pequeñas para el trabajo diario.
- **Flujo de trabajo basado en funciones.** Crear nuevas rama para cada nueva función en la que esté trabajando, de modo que pueda cambiar sin problemas entre ellas y, a continuación, suprimir cada una de las ramas cuando se fusione en su línea principal.
- **Experimentación desechable.** Crear una rama para experimentar, darse cuenta de que no va a funcionar, y solo eliminarlo, abandonar el trabajo.

Una de las facilidades del sistema de control de versiones Git son los submódulos, estos permiten mantener un repositorio Git como una subcarpeta de otro, con lo que será posible clonar un segundo repositorio dentro del que corresponde al proyecto en que se está trabajando, manteniendo separadamente las confirmaciones de cambios en ambos repositorios. Aunque no es objetivo del presente trabajo, se recomienda que en los desarrollos del grupo de investigación se utilice el sistema de submódulos de Git.

1.8.6 Herramienta de modelado

Visual Paradigm para UML 8.0 es una herramienta para desarrollo de aplicaciones utilizando modelado UML ideal para Ingenieros de Software, Analistas de Sistemas y Arquitectos de

sistemas que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos («Visual Paradigm» 2016).

Visual Paradigm también ofrece («Visual Paradigm» 2016):

- Navegación intuitiva entre la escritura del código y su visualización.
- Potente generador de informes en formato PDF/HTML.
- Documentación automática Ad-hoc.
- Ambiente visualmente superior de modelado.
- Sofisticado diagramador automáticamente de layout.
- Sincronización de código fuente en tiempo real.

1.8.7 Lenguaje de modelado

“UML son las siglas de Unificad *Modeling Language* (Lenguaje Unificado de Modelado), notación con que se construyen sistemas por medio de conceptos orientados a objetos” (Larman y Valle 2003).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados (Larman y Valle 2003).

1.9 Consideraciones del capítulo

El capítulo desarrollado contiene los aspectos esenciales relacionados con la fundamentación teórica, como el perfil de la investigación y los estudios realizados sobre las funcionalidades generales de los sistemas para el Diseño Asistido por Computadoras, los diseños arquitectónicos, las metodologías y las tecnologías de desarrollo; el estudio del marco de trabajo OCAF y la comparación con lo implementado en FreeCAD permitió valorar los beneficios y limitaciones de ambos desarrollos y sustentar desde el punto de vista teórico las decisiones de proyecto relacionadas con el tema.

Como resultado del proceso se pudo arribar a las **conclusiones** siguientes:

1. **Que existen características y funcionalidades en los sistemas CAD modernos que son imprescindibles, como las que aseguran la manipulación y edición de objetos geométricos, la parametrización, persistencia de datos y la visualización.**
2. **Que las funcionalidades de los sistemas CAD con carácter más general, pueden estar concentradas en un núcleo, pues son utilizadas por todos los módulos que las integran.**
3. **Que el código fuente disponible de la aplicación FreeCAD, se presentaban como el más apropiado para servir de base en el trabajo de desarrollo.**
4. **Que el estilo que más se ajusta al desarrollo previsto es, en lo fundamental, la Arquitectura por capas**.

2 Propuesta de solución

En el presente capítulo se presenta la propuesta de solución, que incluye el modelo del dominio, los requisitos funcionales y no funcionales, las historias de usuario, la descripción concreta de la propuesta de solución; así como los aspectos de diseño arquitectónico y los patrones orientados a objeto empleados.

2.1 Modelo de dominio

Un modelo del dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés, utilizando la notación UML se representa con un conjunto de diagramas de clases en los que no se define ninguna operación (Larman y Valle 2003). Pueden mostrar:

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

A continuación se presenta el modelo del dominio definido en la presente investigación:

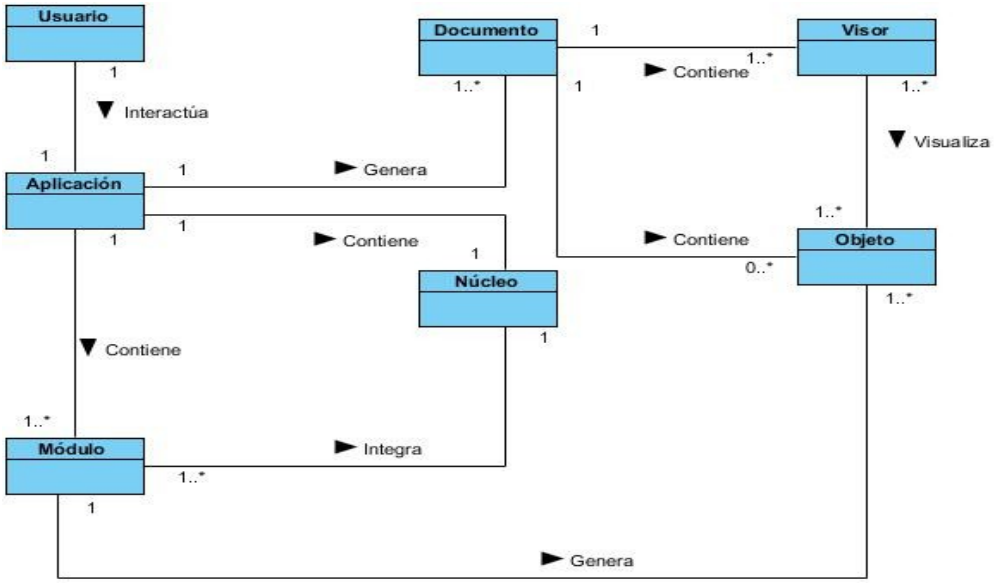

Fig 5: Modelo del dominio

En este punto, es necesario precisar algunos de los principales conceptos empleados en el presente trabajo.

- **Usuario:** Individuo que utiliza una computadora, sistema operativo, servicio o cualquier sistema.
- **Aplicación:** Programa preparado para una utilización específica, como el pago de nóminas, el tratamiento de textos, etcétera («DLE: aplicación» 2017).
- **Documento:** Contenedor que almacenará los objetos y la información relacionada con ellos durante la sesión de trabajo.
- **Módulo:** Parte de la aplicación, con carácter modular, que realizará una o varias de las tareas para la que fue desarrollada.
- **Núcleo:** Sección fundamental de la aplicación, encargada de integrar los módulos y proporcionar las funciones requeridas por los mismos.
- **Visor:** Componente encargado de visualizar los objetos contenidos de un documento, proporcionando múltiples vistas de los mismos.
- **Objeto:** Unidad dentro de la aplicación que almacenará toda la información y características de los elementos a modelar.

El proceso ilustrado en el modelo de dominio debe tratar con una serie de requisitos; sobre estos se realizan las especificaciones necesarias en el apartado siguiente.

2.2 Requisitos

Los requisitos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información (Sommerville 2005). En las líneas siguientes se exponen los requisitos funcionales y no funcionales identificados.

Requisitos funcionales

RF 1. Serialización de los datos.

RF 2. Exportar los Shapes en formato brep.

RF 3. Guardar los datos generados por un proyecto en un único archivo.

RF 4. Restaurar un proyecto.

RF 5. Importar tipos de datos estándares (step, iges, brep).

RF 6. Exportar tipos de datos estándares (step, iges, brep).

RF 7. Visualizar entidades geométricas.

RF 8. Múltiples visores de un documento.

RF 9. Actualización automática de cambios.

RF 10. Vistas ortogonal y perspectiva (ortogonal por defecto).

RF 11. Manipular posición de objetos individuales.

RF 12. Resaltar objetos.

RF 13. Seleccionar objetos.

RF 14. Modificar propiedades de los objetos.

RF 15. Visualizar el árbol de objetos.

Requisitos no funcionales

De portabilidad:

RNF 1. Software: se debe poder instalar en un sistema operativo basado en GNU-Linux de 64bits.

De funcionalidad:

RNF 2. Precisión: debe poseer una alta precisión, en cuanto a la calidad de los datos mostrados.

De usabilidad:

RNF 3. Comprensibilidad: debe poseer imágenes para una mejor comprensión de las opciones.

Para procesar los requisitos funcionales y no funcionales mencionados se elaboran las historias de usuario que se muestran en el apartado que sigue.

2.2.1 Historias de usuarios

Las historias de usuario son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento se pueden modificar, reemplazar por otras más específicas o generales o añadirse nuevas. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (Jeffries, Anderson y Hendrickson 2001). A continuación un ejemplo de historia de usuario.

Número: 3	**Nombre del requisito:** Guardar los datos generados por un proyecto en un único archivo
Programador: Ernesto Alejando Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Media	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir guardar los datos generados por un proyecto en un único archivo. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para guardar los datos generados por un proyecto en un único archivo hay que: - Crear un nuevo documento. - Deben existir objetos dentro del documento. - Los datos deben estar serializados y los shapes exportados en formato brep. **3- Flujo de la acción a realizar:** - El sistema debe permitir guardar los datos generados por un proyecto en un único archivo, esta acción puede realizarse seleccionando la opción Guardar en la interfaz. - El usuario elegirá el nombre del archivo y la dirección donde se guardará. - Si selecciona la opción Cancelar regresará a la vista previa.	
Observaciones:	
Prototipo de interfaz:	

Fig 6: Historia de Usuario "Guardar los datos generados por un proyecto en un único archivo"

Con la información del modelo de dominio, los requisitos y las historias de usuarios se completan los requerimientos para plantear la propuesta de solución.

2.3 Descripción de la propuesta de solución

Para dar solución al problema planteado se había propuesto en el apartado 1.1 el objetivo de **desarrollar un núcleo para aplicaciones de Diseño Asistido por Computadoras, con las funcionalidades generales y básicas que aseguren el modelado paramétrico en dos y tres dimensiones;** en este apartado se ofrece la descripción de la propuesta que permitirá alcanzar la meta planteada.

El núcleo en cuestión será diseñado con las funcionalidades necesarias en correspondencia con los requerimientos ya determinados en el capítulo 1 y que son, en esencia, características y funcionalidades de las aplicaciones CAD; en su composición tendrá una estructura basada en la descripción del apartado 1.2.1, que propone un conjunto de funcionalidades para garantizar la **representación del modelo**, un **subsistema de edición** y un **subsistema de visualización**. Tomando como guía el modelo **Aplicación-Documento-Atributo** proporcionado por OCAF, el núcleo contará con una clase ***CADApplication*** encargada de gestionar los documentos durante la sesión de trabajo.

Los documentos estarán implementados por la clase ***CADDocument,*** encargada de gestionar la serialización de los datos de los objetos, administrar las transacciones y las funciones **Rehacer** y **Deshacer**. Cada documento se almacenará en un único archivo mediante el uso de las funcionalidades brindadas por la clase ***ZipWriter*** que heredará de la clase abstracta ***Writer***; a su vez ***Writer*** será la encargada de gestionar la serialización de los objetos y modelos tridimensionales en formato **BRep**, de los objetos generados durante la sesión de trabajo.

Los datos generados por un documento podrán ser recuperados mediante el uso de la clase *Reader;* cada documento contará con al menos un visor, estos visores serán gestionados mediante la clase ***View3DInventor***, la cual heredará de la clase abstracta ***MDIVIEW*** encargada de gestionar la interacción de las ventanas que contendrán los visores.

La clase ***CADDocumentObject*** servirá de base para todos los objetos generados en los módulos. Estos objetos contarán con un proveedor de vistas, implementado por la clase ***ViewProvider*** responsable de transformar los modelos generados por **OCE**, que luego serán enviados al visor para su posterior visualización mediante ***Coin3D***.

Uno de los requerimientos de la presente propuesta es la posibilidad de poder ser escalada en el futuro, pero manteniendo su estructura relativamente estable en el tiempo, como corresponde a

los núcleos de las aplicaciones; para satisfacer esta exigencia se precisa de una concepción arquitectónica flexible y abierta, que en el siguiente apartado se expone con detalle.

2.4 Diseño arquitectónico del núcleo

Se ha comprendido en el proceso de investigación, la importancia de realizar un diseño arquitectónico apropiado, para sustentar el desarrollo del núcleo de manera coherente y consistente, con el objetivo de lograr un resultado cuyo funcionamiento sea efectivo.

Según (Larman y Valle 2003) el diseño pone énfasis en una solución conceptual orientada a satisfacer los requisitos planteados; se plantea además que el diseño arquitectónico considera el estilo de arquitectura que adoptará el sistema, la estructura y las propiedades de los componentes que lo constituyen y las interrelaciones que ocurren entre sus componentes arquitectónicos (Pressman 2003).

En el esquema de bloques de la siguiente figura se ilustra la conexión del núcleo con los demás componentes de la aplicación.

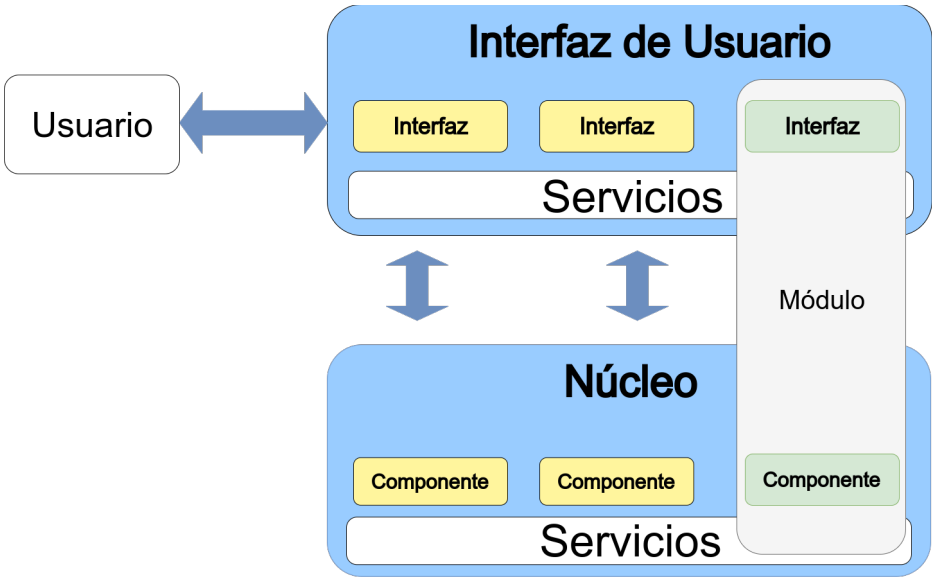

Fig 7: Esquema de la conexión del núcleo con el resto de los componentes de la aplicación

En el esquema se aprecia una estructura compuesta de capas; la capa superior representa la interfaz con la que interactúa el usuario, la cual tiene comunicación con la capa inferior correspondiente al núcleo, que proporciona servicios a todos los componentes de la aplicación; las saetas bidireccionales indican el flujo de comunicación entre los elementos.

Con base en los criterios considerados, durante la investigación, se decidió emplear un diseño arquitectónico por capas en la concepción del núcleo, pero el gran problema de este es la

desventaja señalada en el apartado 1.3, consistente en que la estructuración de los sistemas puede resultar difícil, debido a que los usuarios de las capas superiores, tienen que pasar por las adyacentes para tener acceso a los servicios proporcionados por los niveles inferiores; este inconveniente se puede evitar, permitiendo la comunicación directa entre todas las capas en lugar de usar los servicios proporcionados por las adyacentes (Sommerville 2011).

En la siguiente figura se representa esquemáticamente el diseño conceptual del núcleo.

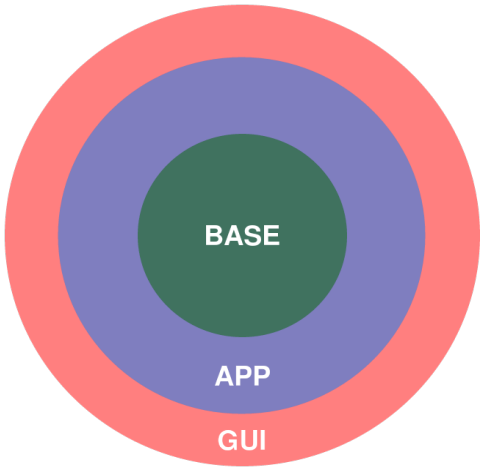

Fig 8: Diseño conceptual del núcleo basado en una arquitectura por capas

La implementación del diseño propuesto requiere el empleo de los patrones de diseño orientados a objeto, estos se exponen en el apartado que sigue.

2.4.1 Patrones de diseño orientados a objeto

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes, en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos (Larman y Valle 2003).

Los patrones GRASP[9] describen los principios fundamentales de la asignación de responsabilidades a objetos. Los definidos en el modelo de clases del núcleo fueron:

- **Experto:** La responsabilidad de la implementación de un método debe recaer sobre la clase que conoce toda la información necesaria para hacerlo (Larman y Valle 2003), este

9 GRASP es un acrónimo de General Responsibility Assignment Software Patterns (patrones generales de software para asignar responsabilidades). El nombre se eligió para sugerir la importancia de aprehender (grasping en inglés) estos principios para diseñar con éxito el software orientado a objetos (Larman y Valle 2003).

patrón se evidencia en las clases ***CADDocumentObject*** y las que hereden de la misma, pues serán las encargadas de conocer como se crea cada objeto.

- **Creador:** Ayuda a identificar quien debe ser el responsable de la creación de nuevos objetos (Larman y Valle 2003). Se evidencia en la clase ***CADDocument*** pues tiene la información necesaria para la creación de objetos.

- **Bajo acoplamiento:** Asigna las responsabilidades a los objetos de modo se reduzca el impacto de los cambios y no incremente el acoplamiento, que es una medida de la fuerza en que las clases se relacionan (Larman y Valle 2003), por ejemplo, la persistencia de los datos, es responsabilidad exclusiva de la clase ***Writer***.

- **Alta cohesión:** Es una medida de cuan relacionadas o enfocadas están las responsabilidades de una clase, en el caso del núcleo se garantizó separando las responsabilidades de los documentos en la clase ***CADDocument***, las de los objetos en la clase ***CADDocumentObject***, entre otros casos (Larman y Valle 2003).

- **Polimorfismo:** Se usa cuando varía el tipo de alternativas o comportamientos relacionados; permite asignar la responsabilidad del comportamiento a los tipos en que varía el mismo (Larman y Valle 2003), se evidencia en las clases ***CADDocumentObject*** y en las que hereden de la misma, implementando el método ***execute***.

La propuesta de solución contendrá los siguientes patrones *GOF*[10]:

- ***Singleton***: Permite asegurar que de una clase concreta existe una única instancia y proporciona un método único que la devuelve (Larman y Valle 2003). Se utilizará en las clases ***CADApplication***, ***SelectionSingleton*** y ***MainWindow***.

- **Observador:** Construye una dependencia entre un sujeto y sus observadores de modo que cada modificación del sujeto sea notificada a los observadores para que puedan actualizar su estado (Larman y Valle 2003). Presente en las clases ***SelectionObserver*** *y* ***TreeView***, entre otras.

10 El nombre de patrones GOF provienen del libro *Design Patterns* (Gamma et al. 1998), debido a que el mismo fue escrito por cuatro autores, estos patrones se conocen como los patrones de la "pandilla de los cuatro" o patrones "GoF" ***(Gang-of-Four)*** (Larman y Valle 2003).

2.5 Consideraciones del capítulo

El resultado fundamental de este capítulo es la definición del diseño arquitectónico del núcleo, para lo cual fue necesario identificar los requisitos con su exposición detallada en las historias de usuario, el estilo arquitectónico y los patrones de diseño orientados a objeto, los que constituyen elementos fundamentales en la etapa de implementación; el modelo de dominio permitió tener una visión de los principales conceptos asociados al núcleo junto con las relaciones que se establecen entre ellos y facilitó la identificación de los aspectos funcionales.

La importancia del diseño arquitectónico del núcleo, reside en que constituye el punto de partida para la fase de Implementación de la solución, en la que se seguirán las pautas establecidas por los patrones GRASP y GOF. En el siguiente capítulo siguiente se expone el contenido de los resultados alcanzados en las fases de Implementación y pruebas de la propuesta planteada.

3 Fases de Implementación y pruebas

En el presente capítulo se abordan cada uno los componentes que integran la etapa de implementación y prueba de la aplicación a desarrollar. Se expone el estándar de codificación, los diseños de los casos de pruebas, así como los resultados obtenidos del proceso de verificación de la calidad.

3.1 Implementación de la propuesta

La fase de implementación del software posee como entradas los artefactos de la fase anterior en la que se expuso el diseño de la propuesta, que contiene el diseño arquitectónico, los patrones de diseño, entre otros elementos. En la implementación se define el estándar de codificación a emplear, se realizan las implementaciones a las historias de usuarios, se define el diagrama de componentes del sistema, entre otras actividades.

3.1.1 Estándar de codificación

Tabla 3: Estándar de Codificación

Descripción	Ejemplo

Todos los nombres de las clases implementadas comenzarán las con clases letra mayúscula. En caso de poseer un nombre compuesto se escribirán de acuerdo a la normativa CamelCase-UpperCamelCase.	class Foo{
	Cuerpo de la clase
	}
	class FooFirst{
	Cuerpo de la clase
	}

Siempre se declara para todas las clases implementadas su respectivo destructor de clase.	virtual ~Foo()
La declaración de funciones o métodos siempre comenzarán en letra inicial minúscula. En caso de ser un nombre compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<Tipo dato retorno> funcion() <Tipo dato retorno> funcionCompuesta() <Tipo dato retorno>funcionDobleCompuesta()
Los atributos siempre estarán escritos con letra minúscula. En caso de ser un nombre	<Tipo dato> atributo;

compuesto se regirá por la normativa CamelCase-lowerCamelCase.	<Tipo dato> atributoNombreCompuesto;
Definición de parámetros dentro de las funciones y constructores de clases	
Los nombres de los identificadores de los parámetros en las funciones deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.	<Tipo dato retorno> funcion(<tipo><id1>, <tipo><id2>, <tipo><idN>)
Los identificadores de los parámetros dentro de los constructores de las clases deben estar escritos con minúscula separados a un espacio cada uno y en caso de ser compuesto utilizar normativa CamelCase-lowerCamelCase.	Clase(<tipo><id1>, <tipo><id2>, <tipo><idN>)
Definición de expresiones	

Las estructuras de control y los bucles estarán definidos de igual manera en ambos casos siguiendo el estándar determinado por el framework de Qt.	Para las estructuras if, else, if else :
	<estructura control>(condición){
	Tarea a ejecutar
	}
	Para los bucles while, for, do while y otros:
	<bucle>(condiciones){
	Tarea a ejecutar
	}

Comentarios en el código según el estándar de C++	
Comentarios pequeños.	/* comentario sencillo */
Otros comentarios.	/* *Comentario */
Comentario de versión, descripción de clase y otras características de la clase o paquete.	/********** *Comentario amplio *

	********** */

3.1.2 Diagrama de componente

Un diagrama de componentes representa las dependencias entre componentes software, incluyendo componentes de código fuente, componentes del código binario, y componentes ejecutables; puede mostrar un sistema configurado con la selección de componentes usados para construirlo o un conjunto de componentes disponibles (una biblioteca de componentes) con sus dependencias.

A continuación se muestra el diagrama de componentes de la solución.

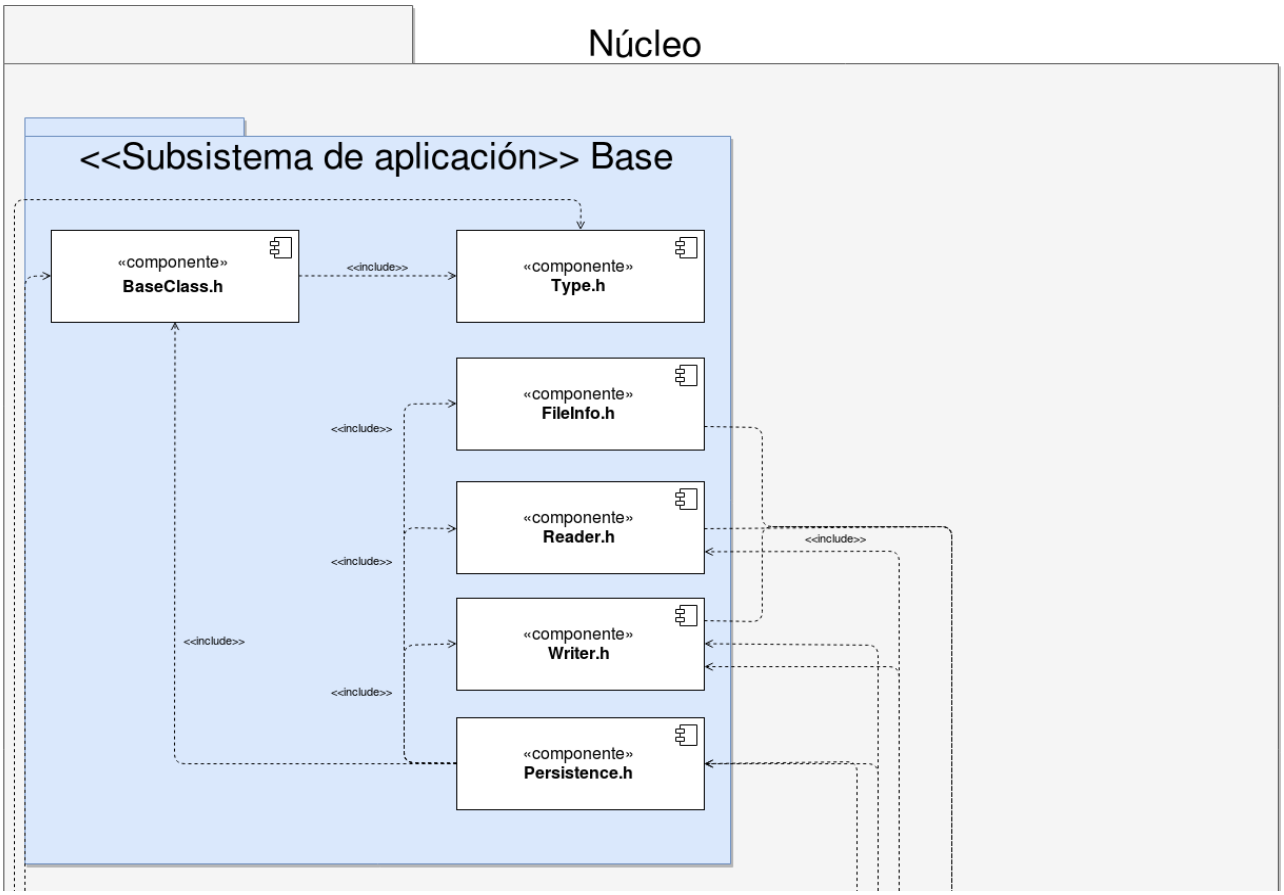

Fig 9: Diagrama de componentes (primera parte)

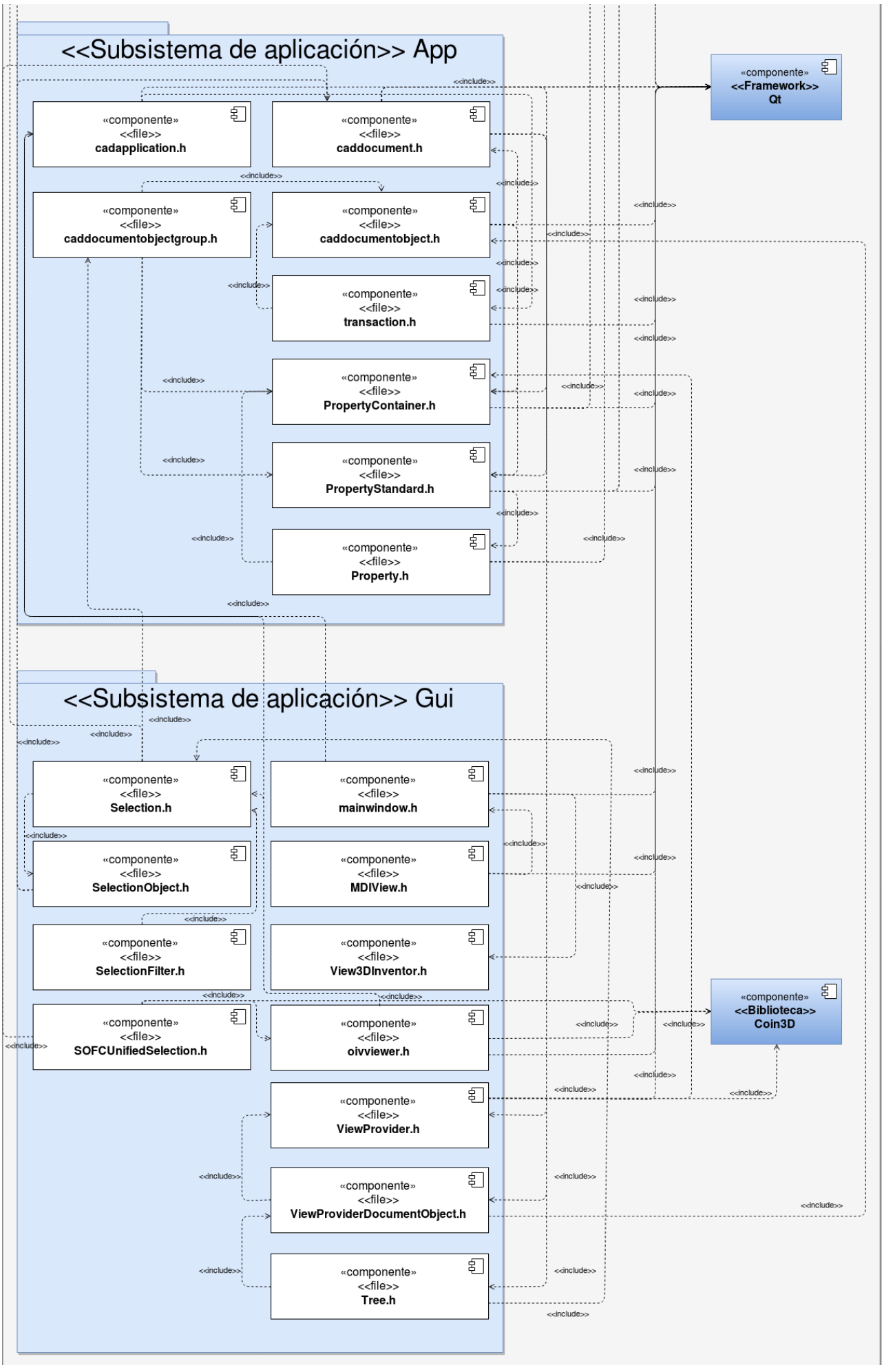

Fig 10: Diagrama de componentes (segunda parte)

Partiendo de la estructura mostrada en el diagrama de componentes expuesto, se procedió a implementar el núcleo, cuyo resultado se resume en el siguiente apartado.

3.1.3 Composición del resultado obtenido en el proceso de implementación

Siguiendo la descripción propuesta en el capítulo anterior, para el desarrollo del núcleo se emplearon los criterios conceptuales manejados durante el proceso de Investigación-Desarrollo, bibliotecas externas y fragmentos del código fuente de la aplicación FreeCAD, que permitieron: la creación de documentos y objetos para su posterior visualización, así como la serialización y persistencia de los datos; en las líneas que siguen se expone en forma resumida la descripción de los módulos y clases principales que componen la solución:

Módulo Base

El módulo ***base*** incluye la mayoría de las funciones y estructuras básicas utilizadas por otros componentes, el mismo contiene las siguientes clases:

Type**:** Almacena la información de las clases proporcionando diversas funcionalidades para trabajar con jerarquías de clase, comparando tipos de clase, creando instancias de objetos mediante sus nombres de clase. Un aspecto importante a tener en cuenta es que su información debe ser registrada antes de que se creen instancias de los objetos.

BaseClass**:** Contiene la implementación de la clase ***BaseClass*** que constituye la raíz del sistema, además contiene las macros[11]:

- TYPESYSTEM_HEADER
- TYPESYSTEM_SOURCE
- TYPESYSTEM_SOURCE_P
- TYPESYSTEM_SOURCE_ABSTRACT
- TYPESYSTEM_SOURCE_ABSTRACT_P

FileInfo**:** Proporciona las funcionalidades para manipular archivos, así como la información de los mismos.

Reader**:** Permite extraer la información de un archivo generado por la aplicación.

11 Una macro es un fragmento de código al que se le ha dado un nombre. Siempre que se utiliza el nombre, se sustituye por el contenido de la macro (GCC team 2017).

Writer*:** contiene las clases ***Writer y ***ZipWriter***, encargadas de garantizar la persistencia de los datos en formato *XML* y un archivo brep por cada figura contenida en el documento, para que sean almacenados en un único archivo.

***Persistence*:** es la interfaz base para las clases que manejen persistencia de los datos.

Módulo App

El módulo ***app*** contiene las clases correspondientes a la capa aplicación y también al *framework* de propiedades.

***Property*:** es la clase padre de todas las propiedades. Las propiedades son objetos que se utilizan en el árbol de documentos para parametrizar los datos, ejemplos: características y su salida gráfica.

PropertyStandard*:** Contiene la implementación de las propiedades básicas del núcleo, algunos ejemplos son: ***PropertyInteger, ***PropertyMap***, ***PropertyFloat***, ***PropertyString*** y ***PropertyBool***.

***PropertyContainer*:** Clase base para todas las clases que contienen propiedades.

***Transaction*:** Representa una transacción atómica del documento mediante la clase *Transaction* y una entrada para un objeto durante una transacción con la clase *TransactionObject*.

***CADDocumentObject*:** Es la representación básica de un objeto dentro de la estructura.

***CADDocumentObjectGroup*:** Permite crear un grupo de objetos *CADDocumentObject*.

***CADDocument*:** Es la representación de una sesión de trabajo dentro del núcleo.

***CADApplication*:** Constituye la raíz de toda la aplicación.

Módulo Gui

El módulo *Gui* contiene las clases correspondientes a la interfaz de usuario y también a los proveedores de vistas y el visor basado en la biblioteca ***Coin3D***.

***Mainwindow.h*:** Contiene la interfaz principal de la aplicación.

***MDIView.h*:** Clase base de todas las ventanas pertenecientes a un documento.

***View3DInventor.h*:** La ventana de vista 3D, contiene al visor *OIVViewer*.

***OIVViewer.h*:** Visor basado en la biblioteca Coin3D.

***ViewProvider.h*:** Mediante esta clase se genera y maneja la visualización de los objetos de la capa App al usuario; esta clase y sus descendientes tienen que ser implementados para cualquier tipo de objeto con el fin de mostrarlos en el *OIVViewer* y *TreeView*.

***ViewProviderDocumentObject.h*:** Provee una interfaz para conectar los objectos representados por la clase *CADDocumentObject*.

***Tree.h*:** Vista del árbol de documentos de la aplicación.

***Selection.h*:** Contiene las clases *SelectionSinglenton* y *SelectionGate*, además de las macros para obtener un observador de selección, *SelectionObserver_HEADER y SelectionObserver_P*.

***SelectionObject.h*:** Permite identificar un objeto (*CADDocumentObject*) en la selección.

***SelectionFilter.h*:** Definición del filtro de selección, permitiendo realizar filtros por caras, vértices o contornos.

***SoFCUnifiedSelection.h*:** Nodo de selección unificada para el visor 3D que eliminará gradualmente todos los nodos de selección de nivel bajo en el proveedor de vistas. El manejo del resaltado y la selección se unificarán aquí.

Tres de los archivos del módulo ***app*** tienen implementadas las funcionalidades principales del sistema de propiedades, estas son ***Property***, ***PropertyStandard*** y ***PropertyContainer***; este sistema se describe en las líneas que siguen.

Sistema de propiedades

Una propiedad es una porción de información, tal como un número o una cadena de texto, que se adjunta a un documento o a un objeto dentro del mismo. Los objetos son nativamente paramétricos, lo que significa que su forma puede basarse en propiedades o incluso depender de otros objetos, todos los cambios se recalculan a petición cada vez que uno de los parámetros ha cambiado (FreeCAD 2016a).

El sistema de propiedades introduce la capacidad de acceder a atributos (variables miembros) de una clase por su nombre sin conocer el tipo de la clase. Funciona como el mecanismo de reflexión de Java o C#. Esta capacidad es introducida por la clase App::PropertyContainer y puede ser utilizada por todas las clases derivadas (FreeCAD 2016a).

Las propiedades pueden ser vistas y modificadas, proporcionando al sistema la capacidad de parametrizar y serializar los objetos, a continuación una lista de las principales propiedades proporcionadas por el núcleo:

- Boolean
- Float
- FloatList
- FloatConstraint
- Angle
- Distance
- Integer
- IntegerConstraint
- Enumeration
- IntegerList
- String
- StringList
- Vector
- VectorList
- Placement
- PlacementLink
- File
- PartShape

Después de haber analizado las clases que integran cada uno de los módulos pertenecientes al núcleo y su sistema de propiedades, se puede exponer la forma en que se realiza la integración de los módulos al núcleo.

3.1.4 Procedimiento para la integración de módulos al núcleo

Una de las propiedades esenciales del núcleo es poder integrar porciones de programas o módulos, este tipo de estructura ofrece ventajas como las que se relacionan a continuación:

- Favorece el trabajo en equipo.
- Disminuye la complejidad del sistema.
- Aumenta la claridad del sistema y de sus módulos.
- Facilita la extensibilidad del sistema, así como las modificaciones y correcciones.

En este apartado se demuestra con un ejemplo la forma en que se integra un módulo al núcleo; en la estructura de este los módulos están contenidos dentro de la carpeta *mod*, la cual contiene una subcarpeta para cada uno de ellos identificada por su nombre; estos están divididos en dos carpetas principales *App* y *Gui* en las que se encuentra distribuida la lógica del negocio y la interfaz gráfica respectivamente.

Para la realización de este ejemplo se crea un módulo *(ModuleBox)* con el objetivo de obtener un cubo cuya geometría está controlada por sus propiedades, en las figuras (11, 12, 13, 14, 15) se muestran la estructura de archivos, métodos, implementación y resultados.

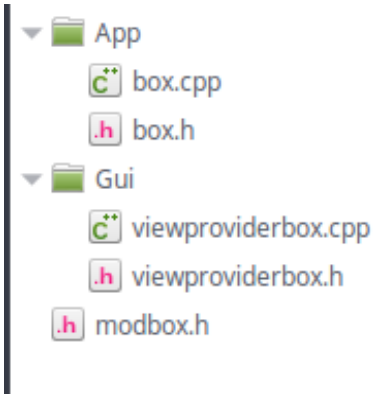

Fig 11: Estructura de archivos del módulo ModuleBox

La geometría de un cubo está definida por tres parámetros: largo, alto y ancho; mediante estos valores podemos obtener un objeto en tres dimensiones con el uso de la función *BrepPrimAPI_MakeBox* proporcionada por *Open CASCADE*. Durante la implementación de la lógica de nuestro módulo se debe heredar de la clase *Part::Feature* y sobrescribir los métodos: *execute* (definiendo como va a ser construido nuestro objeto), *mustExecute* (que propiedades conllevan a rehacer el objeto) y *onChanged* (comportamiento que tendrá el objeto cuando ocurran cambios en sus propiedades). Para acceder a los métodos necesarios para los contenedores de propiedades debemos utilizar las macros *PROPERTY_HEADER* y *PROPERTY_SOURCE*. A continuación la implementación de la clase *Box* (Fig 12, Fig 13).

```
#ifndef BOX_H
#define BOX_H
#include <mod/Part/App/PartFeature.h>
namespace ModuleBox {
class Box : public Part::Feature
{
    /** Macro para la información de tipo durante el tiempo de ejecución y el sistema de
    propiedades, se define utilizando la forma NameSpace::NombreClase  */
    PROPERTY_HEADER(ModuleBox::Box);
public:
    Box();
    //Propiedades que controlan la geometría
    App::PropertyLength Length,Height,Width;
    /** @name Sustitución de métodos */
    //@{
    /// recalcular las características
    App::DocumentObjectExecReturn *execute(void);
    short mustExecute() const;
    /// retorna el nombre del type del ViewProvider
    const char* getViewProviderName(void) const {
        return "ModuleBoxGui::ViewProviderBox";
    }
protected:
    /// llamado por el contenedor cuando una propiedad ha cambiado
    virtual void onChanged(const App::Property* prop);
    //@}
};
} // namespace ModuleBox
#endif // BOX_H
```

Fig 12: Implementación del archivo "box.h"

```
#include "box.h"
#include <Precision.hxx>
#include <BRepPrimAPI_MakeBox.hxx>

PROPERTY_SOURCE(ModuleBox::Box, Part::Feature)

namespace ModuleBox {

   Box::Box()
   {
      ADD_PROPERTY_TYPE(Length,(10.0f),"Box",App::Prop_None,"The length of the box");
      ADD_PROPERTY_TYPE(Width ,(10.0f),"Box",App::Prop_None,"The width of the box");
      ADD_PROPERTY_TYPE(Height,(10.0f),"Box",App::Prop_None,"The height of the box");
   }
   short Box::mustExecute() const
   {
      if (Length.isTouched() || Height.isTouched() || Width.isTouched() )
         return 1;
      return Feature::mustExecute();
   }
   App::DocumentObjectExecReturn *Box::execute(void)
   {
      double L = Length.getValue();
      double W = Width.getValue();
      double H = Height.getValue();
      if (L < Precision::Confusion())
         return new App::DocumentObjectExecReturn("Length of box too small");
      if (W < Precision::Confusion())
         return new App::DocumentObjectExecReturn("Width of box too small");
      if (H < Precision::Confusion())
         return new App::DocumentObjectExecReturn("Height of box too small");

      try {
         // Construir el cubo mediante la API Opencascade con los valores de las propiedades
         BRepPrimAPI_MakeBox mkBox(L, W, H);
         TopoDS_Shape ResultShape = mkBox.Shape();
         this->Shape.setValue(ResultShape);
      }
      catch (Standard_Failure) {
         Handle_Standard_Failure e = Standard_Failure::Caught();
         return new App::DocumentObjectExecReturn(e->GetMessageString());
      }
      return App::CADDocumentObject::StdReturn;
   }
   void Box::onChanged(const App::Property* prop)
   {
      if (prop == &Length || prop == &Width || prop == &Height) {
         if (!isRestoring()) {
            App::DocumentObjectExecReturn *ret = recompute();
            delete ret;
         }
      }
      Part::Feature::onChanged(prop);
   }
} // namespace ModuleBox
```

Fig 13: Implementación del archivo "box.cpp"

Después de haber implementado la lógica de nuestro módulo en los archivos ***box.h*** (Fig 12) y ***box.cpp*** (Fig 13), es necesario realizar un proveedor de vista *(**ViewProvider**)* para traducir la salida de la *API Open CASCADE* y lograr su posterior visualización mediante la biblioteca ***Coin3D***.

Nuestro proveedor de vistas debe heredar de la clase abstracta ***PartGui::ViewProviderPart*** y definir el método ***getDisplayModes*** necesario para establecer los tipos de visualización con los que puede contar nuestro objeto; en la implementación del presente trabajo se definen cuatro variantes de visualización: *Flat Lines* la cual presenta la geometría, vértices y líneas de contorno; *Shaded* solo exhibe la geometría*; Wireframe* muestra los vértices y líneas de contorno y *Points* expone los vértices. Las siguientes figuras muestran la implementación del proveedor de vistas ***ViewProviderBox***.

```
#ifndef VIEWPROVIDERBOX_H
#define VIEWPROVIDERBOX_H

#include "ui/ViewProviders/ViewProviderPart.h"

namespace ModuleBoxGui {

class ViewProviderBox: public PartGui::ViewProviderPart
{
   PROPERTY_HEADER(ModuleBoxGui::ViewProviderBox);

public:
   /// constructor
   ViewProviderBox();
   /// destructor
   virtual ~ViewProviderBox();

   QVector<QString> getDisplayModes(void) const;
};

} // namespace ModuleBoxGui

#endif // VIEWPROVIDERBOX_H
```

Fig 14: Implementación del archivo "viewproviderbox.h"

```
#include "viewproviderbox.h"

namespace ModuleBoxGui {

//**************************************************************************
// Construction/Destruction
// macro para obtener la implementación las funcionalidades de las propiedades
PROPERTY_SOURCE(ModuleBoxGui::ViewProviderBox,PartGui::ViewProviderPart)


ViewProviderBox::ViewProviderBox()
{

}

ViewProviderBox::~ViewProviderBox()
{

}
//
**************************************************************************
*******

QVector<QString> ViewProviderBox::getDisplayModes(void) const
{
   // get the modes of the father
   QVector<QString> StrList;

   // add your own modes
   StrList.push_back("Flat Lines");
   StrList.push_back("Shaded");
   StrList.push_back("Wireframe");
   StrList.push_back("Points");

   return StrList;
}
} // namespace ModuleBoxGui
```

Fig 15: Implementación del archivo "viewproviderbox.cpp"

Una vez implementadas las clases y métodos necesarios, se debe agregar la inicialización de las clases mediante el uso del método *init* en el archivo ***modbox.h*** e implementar la llamada a la creación del objeto del módulo.

```
#ifndef MODBOX_H
#define MODBOX_H
#include <mod/ModuleBox/App/box.h>
#include <mod/ModuleBox/Gui/viewproviderbox.h>

inline static void initTypes(){
   ModuleBox::Box                    ::init();
   ModuleBoxGui::ViewProviderBox     ::init();
}

#endif // MODBOX_H
```

Fig 16: Implementación del archivo modbox.h

```
Part::Feature *shp = static_cast<Part::Feature*>(getApplication()→activeDocument()→
addObject("ModuleBox::Box", "ModuleBox"));
shp→execute();
```

Fig 17: Implementación la llamada a la creación del objeto del módulo

Concluidos los pasos anteriormente expuestos se logra obtener el siguiente resultado:

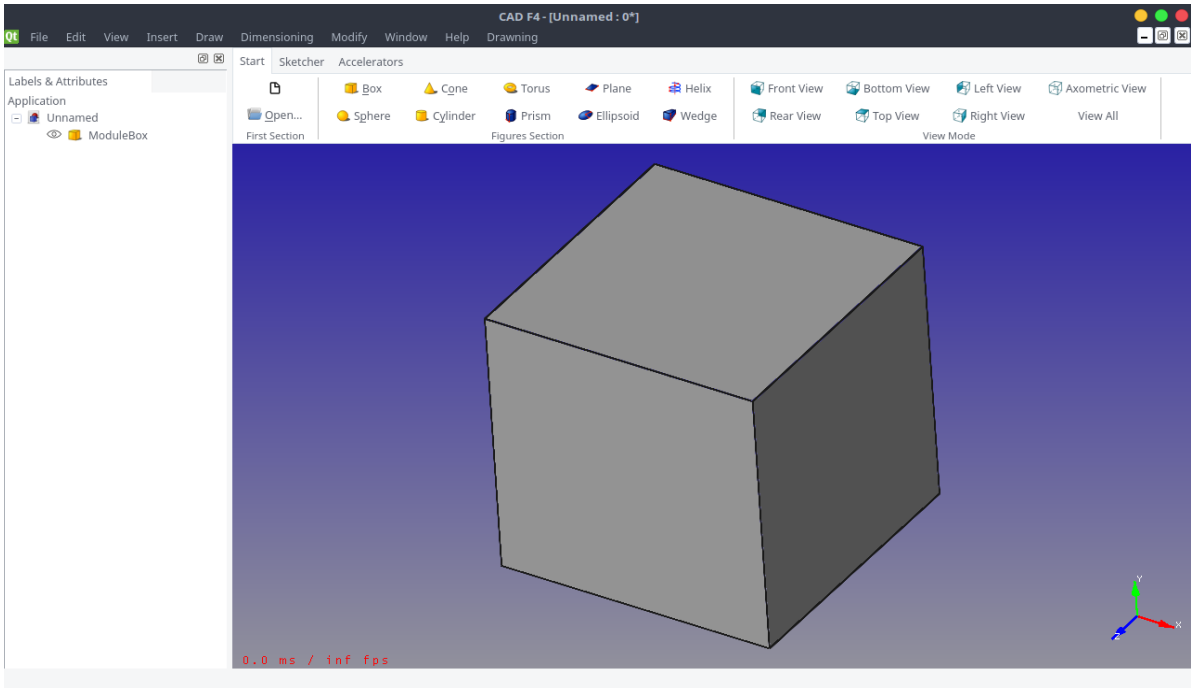

Fig 18: Resultado de la ejecución del módulo ModuleBox

El procedimiento de integración de los módulos que ha sido descrito, requiere que se compile la aplicación con los módulos incluidos, lo que tiene como desventaja que para agregar un nuevo módulo es necesario recompilar la aplicación; este problema puede tener varias formas de

solución, por ejemplo, en la aplicación FreeCAD se integran módulos mediante funciones en Python y bibliotecas compiladas en lenguaje C++. La proyección que se prevé **implementar en los desarrollos del grupo de investigación, es establecer una estructura para la compilación de módulos como bibliotecas, lo que permitiría integrar módulos sin necesidad de recompilar la aplicación**.

Luego de finalizar el proceso de integración de los módulos al sistema se hace necesario comprobar el correcto funcionamiento del mismo mediante la aplicación de pruebas; el proceso de realización y resultados de las mismas se exponen en el siguiente epígrafe.

3.2 Pruebas

Durante las fases anteriores de definición y de desarrollo, se intenta construir el software partiendo de un concepto abstracto y llegando a una implementación tangible. A continuación, llegan las pruebas donde se crean una serie de casos de prueba que intentan destruir el software construido. De hecho, según (Pressman 2006) "...las pruebas son uno de los pasos de la ingeniería del software que se puede ver como destructivo en lugar de constructivo...".

Ian Sommerville en su libro Ingeniería del software plantea que el proceso de prueba de software tiene dos objetivos distintos (Sommerville 2005):

1. Demostrar al desarrollador y al cliente que el software satisface sus requisitos. Esto significa que debería haber al menos una prueba para cada requerimiento o característica que se incorporará a la entrega del producto.

2. Descubrir defectos en el software en el que el comportamiento de este es incorrecto, no deseable o no cumple su especificación. La prueba de defectos está relacionada con la eliminación de todos los tipos de comportamientos del sistema no deseables, tales como caídas del sistema, interacciones no permitidas con otros sistemas, cálculos incorrectos y corrupción de datos.

La metodología AUP-UCI define tres etapas para la realización de las pruebas (Sánchez 2015):

- **Pruebas internas:** se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posible componentes de pruebas ejecutables para automatizar las pruebas.

- **Pruebas de liberación:** Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables[12] de los proyectos antes de ser entregados al cliente para su aceptación.
- **Pruebas de Aceptación:** Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

Métodos de prueba

Las pruebas en sí mismas deben mostrar un conjunto de características que logren la meta de encontrar la mayor cantidad de errores con el mínimo esfuerzo. Para llevar a cabo este objetivo, se emplearán los dos métodos de prueba (Pressman 2006):

- **Prueba de caja blanca:** se basa en el examen cercano de los detalles de procedimiento. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que:
  - Garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez.
  - Revisen todas las decisiones lógicas en sus lados, verdadero y falso.
  - Ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas.
  - Revisen estructuras de datos internas para garantizar su validez.
- **Prueba de caja negra:** se refiere a las pruebas que se llevan a cabo en la interfaz del software, examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del software. Intentan encontrar errores en las siguientes categorías:
  - Funciones incorrectas o faltantes.
  - Errores de interfaz.
  - Errores en las estructuras de datos o en el acceso a bases de datos externas.
  - Errores de comportamiento o rendimiento.

12 Los productos entregables son aquellos que se entregan oficialmente al cliente como parte del desarrollo en fechas previamente acordadas (Toro y Jiménez 2000).

- Errores de inicialización y terminación.

Para validar el correcto funcionamiento del núcleo se tienen las pruebas unitarias, las pruebas funcionales y las pruebas de aceptación realizadas por el cliente.

Pruebas Unitarias

Para la realización de las pruebas unitarias se utiliza el Qt Test, un *framework* para realizar pruebas unitarias a aplicaciones y bibliotecas basadas en Qt. Qt Test Proporciona todas las funcionalidades comúnmente encontradas en los framework de pruebas, así como extensiones para probar interfaces gráficas de usuario («Qt Test Overview | Qt Test 5.8» 2017). Los resultados de la ejecución de las pruebas se muestran en (Fig 19).

```
********* Start testing of PruebasNucleo *********
Config: Using QtTest library 5.5.1, Qt 5.5.1 (x86_64-l
PASS   : PruebasNucleo::initTestCase()
PASS   : PruebasNucleo::testCADDocument()
PASS   : PruebasNucleo::testCADDocumentObject()
PASS   : PruebasNucleo::testViewProviders()
PASS   : PruebasNucleo::testSave()
PASS   : PruebasNucleo::testRestore()
PASS   : PruebasNucleo::testCreateObject()
PASS   : PruebasNucleo::testDeleteObject()
PASS   : PruebasNucleo::testChangeObject()
PASS   : PruebasNucleo::testCreateViewers()
PASS   : PruebasNucleo::testCloseViewers()
PASS   : PruebasNucleo::testChangeCameraType()
PASS   : PruebasNucleo::testSerialize()
PASS   : PruebasNucleo::testExport()
PASS   : PruebasNucleo::testImport()
PASS   : PruebasNucleo::testSelection()
PASS   : PruebasNucleo::testTreeView()
PASS   : PruebasNucleo::cleanupTestCase()
Totals: 18 passed, 0 failed, 0 skipped, 0 blacklisted
********* Finished testing of PruebasNucleo *********
```

Fig 19: Ejecución de las pruebas

Pruebas funcionales

Para validar los requisitos funcionales de la aplicación se diseñan casos de prueba a partir de las historias de usuario con el objetivo fundamental de encontrar la mayor cantidad posible de deficiencias existentes en las funcionalidades implementadas. A continuación se presenta el Caso de Prueba de la Historia de usuario “Guardar los datos generados por un proyecto en un único archivo”.

Tabla 3: Caso de Prueba de la Historia de usuario “Guardar los datos generados por un proyecto en un único archivo”

Descripción General			
Permitir guardar los datos generados por un proyecto en un único archivo			
SC 1 Guardar los datos generados por un proyecto			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Opción de guardar un proyecto.	Selecciona la opción de guardar en el menú de la aplicación y se muestra una ventana para seleccionar el nombre y ubicación donde se almacenará el documento.	Brinda la posibilidad de seleccionar la ubicación donde se guardará el proyecto y poder establecer su nombre. Permite además: - Cancelar la operación en cualquier momento.	Menú/ Save
EC 1.2 Opción de cancelar.	Selecciona la opción de Cancelar.	Elimina los datos creados. Cierra la ventana.	Menú/ Save/ Cancel

Resultado de las pruebas

La realización de pruebas funcionales permitió identificar la presencia de no conformidades, las cuales se listan a continuación:

Tabla 4: No conformidades detectadas en las pruebas.

No. NC	Requisito Funcional	Descripción	Complejidad	Estado
1	RF 1	No se generan los identificadores universales de los documentos.	Media	Resuelta
2	RF 2	Se genera un solo archivo temporal, provocando que los archivos brep de los objetos se sobrescriban.	Media	Resuelta
3	RF 3	No se establecen correctamente los permisos a los archivos dentro del comprimido.	Alta	Resuelta
4	RF 4	No se recupera el nombre del documento correctamente.	Media	Resuelta
5	RF 7	Se visualizan los objetos dobles en el visor.	Baja	Resuelta
6	RF 10	Se desactiva la cámara ortogonal por defecto.	Baja	Resuelta
7	RF 13	No se resaltan los objetos cuando colocas el cursor sobre ellos.	Alta	Resuelta
8	RF 16	Cuando se selecciona un objeto desde el árbol no permite posteriormente seleccionar otro.	Media	Resuelta
9	RF 8	No se activa el documento cuando se realiza un cambio	Media	Resuelta

		entre múltiples visores de varios documentos, provocando que las figuras se creen en un documento no deseado.		
10	RF 11	No permite cambiar hacía vista perspectiva.	Baja	Resuelta

En el siguiente gráfico se muestra un resumen correspondiente al proceso de pruebas:

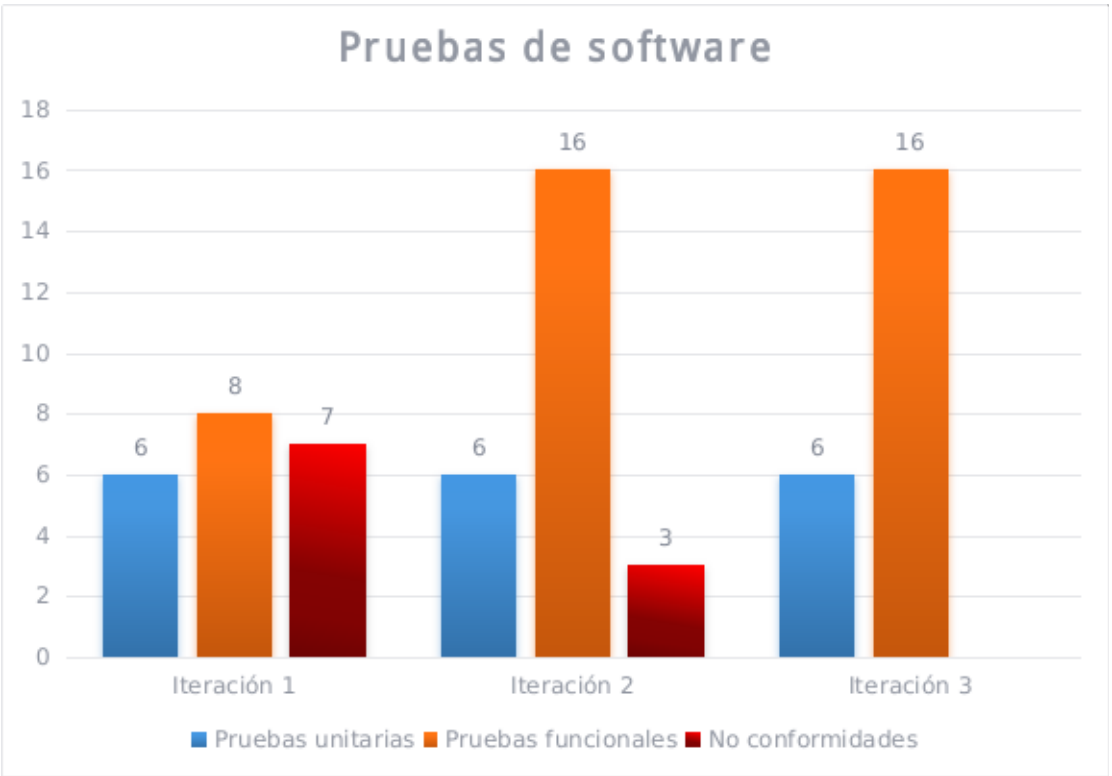

Fig 20: Gráfico de iteraciones de las pruebas realizadas.

Pruebas de integración

Las pruebas de integración son una extensión lógica de las pruebas unitarias. El proceso de integración del sistema implica construir este a partir de sus componentes y probar el sistema resultante para encontrar problemas que pueden surgir debido a la integración de los componentes (Abner Gerardo 2013). A continuación le muestra la evidencia de la exitosa integración de los módulos desarrollados por el grupo SIPII al núcleo.

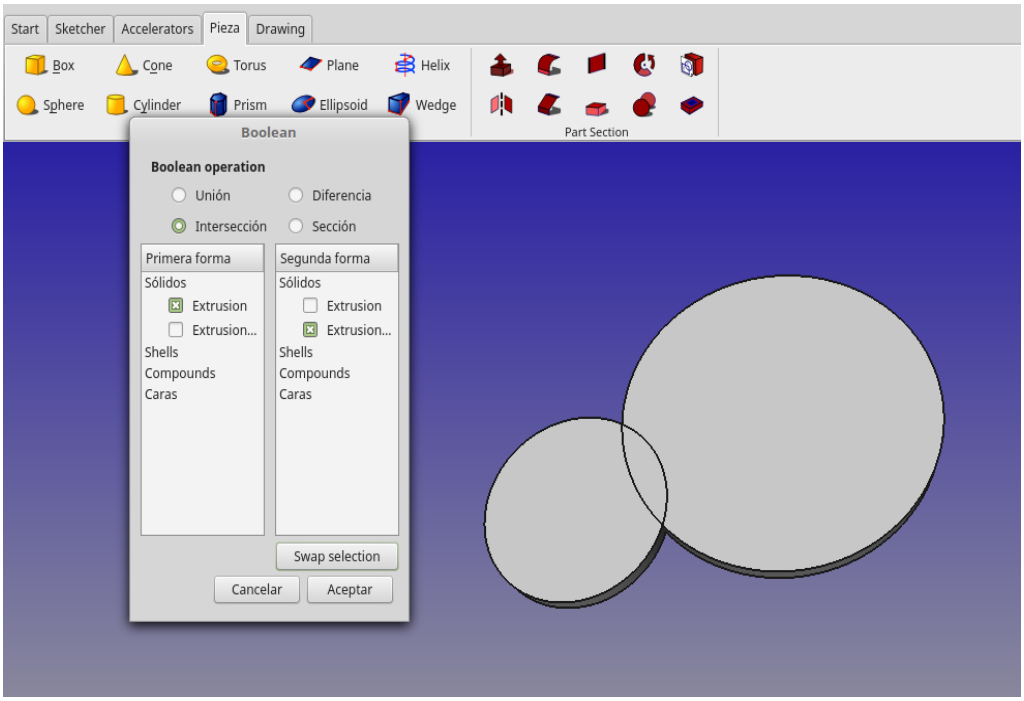

Fig 21: Módulo Part

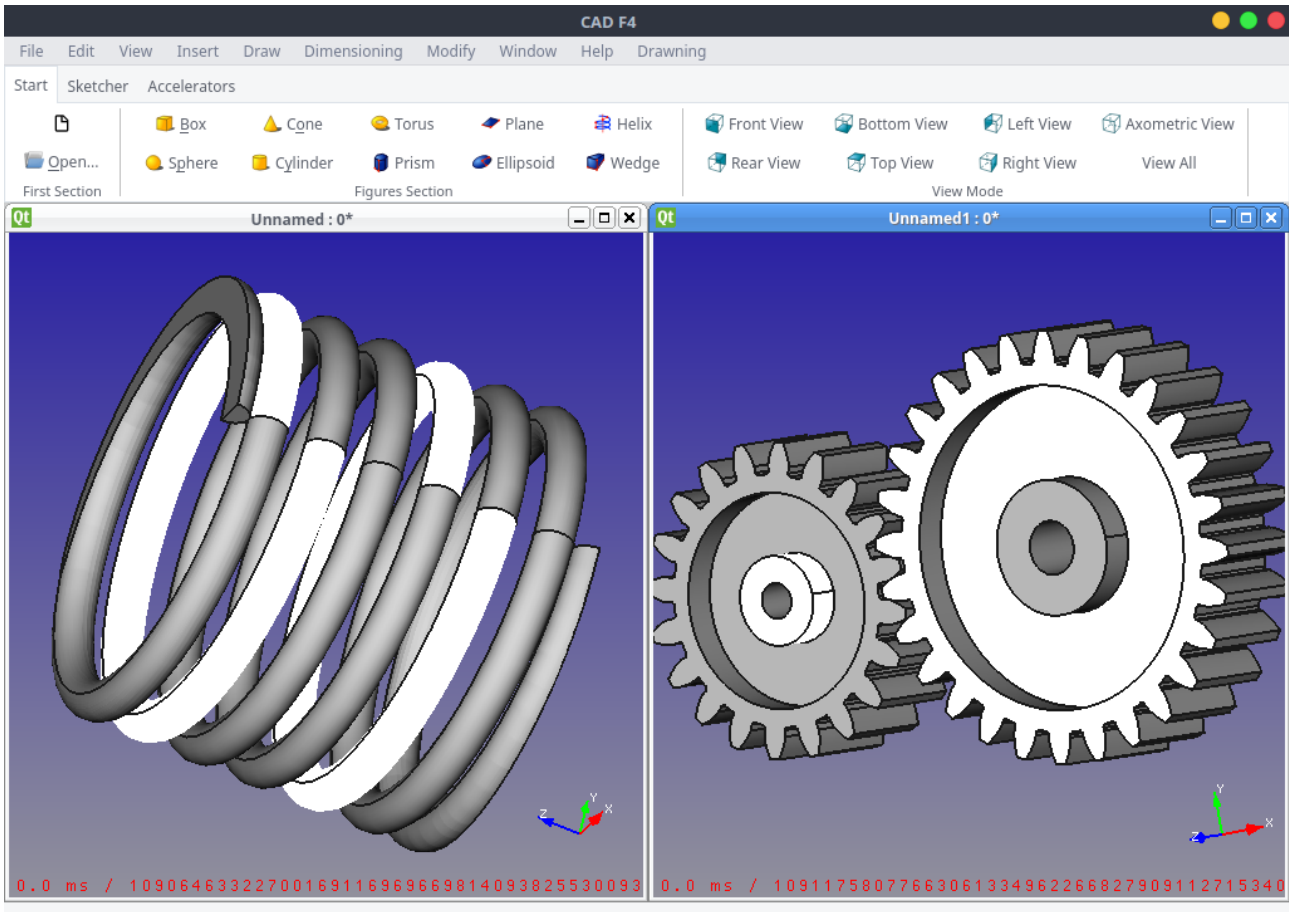

Fig 22: Módulos Springs y SpurGear.

3.3 Consideraciones del capítulo

Los resultados principales del capítulo están conformados por la **descripción de la composición del núcleo implementado** y el **procedimiento para integrar módulos**; la validación de estos resultados está avalada por los resultados de las pruebas realizadas, descritas también en el cuerpo del capítulo.

Se puede remarcar que la importancia de los resultados obtenidos en el capítulo representan el **cumplimiento del objetivo planteado y la solución al problema formulado** en el apartado 1.1.

4 Conclusiones

Como resultado del proceso de Investigación-Desarrollo realizado se arribó a las siguientes conclusiones generales:

- **Las funcionalidades de los sistemas CAD con carácter más general, pueden estar concentradas en un núcleo y que el código fuente de la aplicación FreeCAD ofrecía un basamento apropiado para obtener la solución.**
- **El diseño arquitectónico más apropiado para el desarrollo de un núcleo de aplicaciones para el Diseño Asistido por Computadora debe basarse, en lo fundamental, en una Arquitectura por capas que permita la comunicación directa entre todas sus capas.**
- **La eficacia del resultado obtenido, como núcleo destinado a sistemas de Diseño Asistido por Computadoras, está avalada por los resultados de las pruebas realizadas y garantiza la integración de los módulos existentes.**

5 Recomendaciones

A partir de los resultados obtenidos se recomienda:

- **Implementar también visores de OCC, Vtk y OpenGL para evaluar los niveles de efectividad y de eficiencia en los desarrollos que se ejecutan.**
- **Establecer una estructura para la compilación de módulos como bibliotecas, lo que permitiría integrar módulos sin necesidad de recompilar la aplicación.**
- **Utilizar el sistema de submódulos de Git, para facilitar el trabajo de desarrollo manteniendo separadas las confirmaciones de cambios en diferentes repositorios.**

6 Bibliografía

1. ABNER GERARDO, 2013. Pruebas de sistemas y aceptacion. [en línea]. Educación. S.l. Disponible en: https://es.slideshare.net/abnergerardo/pruebas-de-sistemas-y-aceptacion-23663195.

2. About Qt. [en línea], 2016. [Consulta: 25 enero 2017]. Disponible en: http://wiki.qt.io/About_Qt.

3. AMADOR, A.L.O., 2015. *Aplicación para visualizar entidades geométricas, topologías y modelos de componentes mecánicos (CAD2D)*. Ingeniería en Ciencias Informáticas. Cuba: Universidad de las Ciencias Informáticas.

4. BAYERISCHER RUNDFUNK, 2016. *Autocad Mechanical 2017 Parameters, Geometric and Dimentional Constrains* [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.youtube.com/watch?v=cVE7WT01avI.

5. BETTIG, B. y HOFFMANN, C.M., 2011. Geometric constraint solving in parametric computer-aided design. *Journal of computing and information science in engineering*, vol. 11, no. 2, pp. 021001.

6. BRÜDERLIN, B. y ROLLER, D., 2012. *Geometric Constraint Solving and Applications*. S.l.: Springer Science & Business Media. ISBN 978-3-642-58898-3.

7. CADPOINTDIRECT, 2017. *Autodesk Inventor - Features - Parametric Modeling* [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.youtube.com/watch?v=Bja-30GWtHo.

8. CATIA, 2017. CATIA. [en línea]. [Consulta: 14 junio 2017]. Disponible en: https://www.3ds.com/products-services/catia/.

9. CODE-ASTER, 2017. Code_Aster - Code_Aster. [en línea]. [Consulta: 15 junio 2017]. Disponible en: http://www.code-aster.org/spip.php?article272.

10. CODE-SATURNE, 2017. Code_Saturne. [en línea]. [Consulta: 15 junio 2017]. Disponible en: http://code-saturne.org/cms/.

11. Coin3D. [en línea], 2016. [Consulta: 25 enero 2017]. Disponible en: https://bitbucket.org/Coin3D/.

12. DASSAULT SYSTÈMES, 2017. SolidWorks. [en línea]. [Consulta: 14 junio 2017]. Disponible en: http://www.solidworks.es/.

13. DIE TECHNICA, 2017. *02 CATIA V5 User Interface Parametric Modeling* [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.youtube.com/watch?v=zTMbWM7e_Wo.

14. DLE: aplicación. [en línea], 2017. [Consulta: 17 marzo 2017]. Disponible en: http://dle.rae.es/?id=3CdjxNg.

15. DRUCKER, J., 2011. Humanities approaches to graphical display. *Digital Humanities Quarterly*, vol. 5, no. 1.

16. ELMER, 2017. CSC - Elmer. [en línea]. [Consulta: 15 junio 2017]. Disponible en: https://www.csc.fi/web/elmer.

17. FERNÁNDEZ FERRER, A., 2015. Componente para la modelación de árboles escalonados en el Sistema CAD 2D. [en línea], [Consulta: 28 junio 2017]. Disponible en: http://repositorio.uci.cu/jspui/handle/123456789/7378.

18. FREECAD, 2016a. Manual:Introduction - FreeCAD Documentation. [en línea]. [Consulta: 31 mayo 2017]. Disponible en: https://www.freecadweb.org/wiki/Manual:Introduction.

19. FREECAD, 2016b. Third Party Libraries - FreeCAD Documentation. [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.freecadweb.org/wiki/Third_Party_Libraries#Coin3D.

20. FREECAD, 2017. History - FreeCAD Documentation. [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.freecadweb.org/wiki/History.

21. GAMMA, E., HELM, R., JOHNSON, R. y VLISSIDES, J., 1998. *Design Patterns CD: Elements of Reusable Object-oriented Software*. S.l.: Addison-Wesley. ISBN 978-0-201-63498-3.

22. GCC TEAM, 2017. The C Preprocessor: Macros. *GCC online documentation* [en línea]. [Consulta: 14 junio 2017]. Disponible en: https://gcc.gnu.org/onlinedocs/cpp/Macros.html.

23. GONZÁLEZ, G.G., 2016. *Componente para acelerar el diseño de resortes helicoidales*. Ingeniería en Ciencias Informáticas. Cuba: Universidad de las Ciencias Informáticas.

24. GONZÁLEZ, J.C.P., 2016. *Componente para el modelado de vigas para el Sistema CAD 2D*. Ingeniería en Ciencias Informáticas. Cuba: Universidad de las Ciencias Informáticas.

25. HOWENGINEERSDOIT!, 2013. *Catia V5 Tutorials|Part Design|Surface Based Feature|Close Surface* [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.youtube.com/watch?v=R3II3kPKV6U.

26. JEFFRIES, R., ANDERSON, A. y HENDRICKSON, C., 2001. *Extreme Programming Installed*. S.l.: Addison-Wesley Professional. ISBN 978-0-201-70842-4.

27. LARMAN, C. y VALLE, B.M., 2003. *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado* [en línea]. S.l.: Pearson Educación. Fuera de colección Out of series. ISBN 978-84-205-3438-1. Disponible en: https://books.google.es/books?id=3NEaPwAACAAJ.

28. LIGERO, J.A., 2008. *Fundamentos del KBE. Aplicación al diseño de engranajes de ejes paralelos con Catia v5*. Ingeniería Gráfica. Sevilla: Escuela Superior de Ingenieros.

29. MARTÍNEZ, R.R., 2004. Criterios para Seleccionar Sistemas de Diseño y Manufactura Asistidos por Computadora (CAD/CAM). *Información tecnológica*, vol. 15, no. 2, pp. 91-94. ISSN 0718-0764. DOI 10.4067/S0718-07642004000200016.

30. MAYER, W., 2010. Possibility of replacing Open CASCADE - FreeCAD Forum. [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://forum.freecadweb.org/viewtopic.php?t=344.

31. MONTANO, J.L.M. de O., 2015. La migración hacia software libre en Cuba:complejo conjunto de factores sociales y tecnológicos en el camino de la soberanía nacional. *Revista Universidad y Sociedad*, vol. 7, pp. 119-125.

32. OCAF, O., 2016. Open CASCADE Technology: OCAF. [en línea]. [Consulta: 25 enero 2017]. Disponible en: https://www.opencascade.com/doc/occt-7.1.0/overview/html/occt_user_guides__ocaf.html.

33. OCTT: Overview. [en línea], 2017. [Consulta: 25 enero 2017]. Disponible en: https://dev.opencascade.org/doc/overview/html/index.html.

34. OPEN CASCADE S.A.S, 2013. Open CASCADE Technology: BRep Format. [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.opencascade.com/doc/occt-6.7.0/overview/html/occt_brep_format.html.

35. OPEN CASCADE S.A.S, 2017. Company | OPEN CASCADE. [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.opencascade.com/content/company.

36. Open CASCADE Technology: OCAF. [en línea], 2016. [Consulta: 25 enero 2017]. Disponible en: https://www.opencascade.com/doc/occt-7.1.0/overview/html/occt_user_guides__ocaf.html.

37. OPENGL, 2016. OpenGL Overview. [en línea]. [Consulta: 22 enero 2017]. Disponible en: https://www.opengl.org/about/#1.

38. OPENSCENEGRAPH, 2016. OpenSceneGraph. [en línea]. [Consulta: 25 enero 2017]. Disponible en: http://www.openscenegraph.org/.

39. PCB-3D, 2017. STEP file FAQ – PCB 3D. [en línea]. [Consulta: 28 junio 2017]. Disponible en: https://www.pcb-3d.com/knowledge-base/step-file-faq/.

40. PEÑA PEÑATE, A., 2015. AsiXMec. [en línea]. [Consulta: 14 junio 2017]. Disponible en: http://www.3dcadportal.com/asixmec-aplicacion-cubana-para-el-diseno-e-ingenieria-asistida-por-computadora.html.

41. POBLET, J.M., 1986. *Sistemas CAD, CAM, CAE: Diseño y fabricación por computador* [en línea]. S.l.: Marcombo Boixareu. Mundo electrónico. ISBN 978-84-267-0621-8. Disponible en: https://books.google.com.cu/books?id=ffnmPAAACAAJ.

42. PRESSMAN, R.S., 2003. *Ingeniería del software: un enfoque práctico*. S.l.: McGraw-Hill. ISBN 978-84-481-3214-9.

43. PRESSMAN, R.S., 2006. *Ingeniería del software: un enfoque práctico*. S.l.: McGraw-Hill. ISBN 978-970-10-5473-4.

44. Qt Test Overview | Qt Test 5.8. *Qt Test Overview | Qt Test 5.8* [en línea], 2017. [Consulta: 15 mayo 2017]. Disponible en: http://doc.qt.io/qt-5/qtest-overview.html.

45. RIBARSKY, W. y FOLEY, J.D., 1994. *Next-generation Data Visualization Tools*. S.l.: Graphics, Visualization & Usability Center, Georgia Institute of Technology.

46. ROJAS, O.R.L. y L.R., 2014. Diseño asistido por computador. *Industrial Data*, vol. 9, no. 1, pp. 007–015. ISSN 1810-9993.

47. SÁNCHEZ, T.R., 2015. *Metodología de desarrollo para la Actividad productiva de la UCI.* 2015. S.l.: Universidad de las Ciencias Informáticas.

48. SANTOS, S.G., 2015. *Componente para la modelación de engranajes cilíndricos de dientes rectos*. Ingeniería en Ciencias Informáticas. Cuba: Universidad de las Ciencias Informáticas.

49. SLYADNEV, S., 2014. isicad :: Open CASCADE Technology Overview. [en línea]. [Consulta: 28 junio 2017]. Disponible en: http://isicad.net/articles.php?article_num=17368.

50. SOFTWARE-ENGINEERING-BOOK, 2015. Ch6 architectural design. [en línea]. Technology. S.l. [Consulta: 23 junio 2017]. Disponible en: https://www.slideshare.net/software-engineering-book/ch6-architectural-design.

51. SOMMERVILLE, I., 2005. *Ingeniería del software*. S.l.: Pearson Educación. ISBN 978-84-7829-074-1.

52. SOMMERVILLE, I., 2011. *Ingeniería de software*. S.l.: Pearson Educación de México. ISBN 978-607-32-0603-7.

53. SPITZ, S. y RAPPOPORT, A., 2004. Integrated feature-based and geometric CAD data exchange. *Proceedings of the ninth ACM symposium on Solid modeling and applications*. S.l.: Eurographics Association, pp. 183–190.

54. STROUSTRUP, B., 2013. *The C++ Programming Language*. S.l.: Addison-Wesley. ISBN 978-0-13-352285-3.

55. TORO, A.D. y JIMÉNEZ, B.B., 2000. Metodología para la elicitación de requisitos de sistemas software. *Informe Técnico LSI-2000-10. Facultad de Informática y Estadística Universidad de Sevilla*,

56. TORRES, J.C., 2005. Diseño asistido por ordenador. [en línea]. Disponible en: http://lsi.ugr.es/~cad/teoria/Tema1/RESUMENTEMA1.PDF.

57. TORVALDS, L. y HAMANO, J., 2010. Git: Fast version control system. *http://git-scm.com*,

58. Visual Paradigm. [en línea], 2016. [Consulta: 25 enero 2017]. Disponible en: https://www.visual-paradigm.com/features/.

59. VTK, 2016. VTK. [en línea]. [Consulta: 25 enero 2017]. Disponible en: http://www.vtk.org/overview/.

60. WILSON, P.R., 1987. A short history of CAD data transfer standards. *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 64–67.

61. WIPO, 2017. Preguntas frecuentes: los diseños industriales (también denominados dibujos y modelos industriales). *World Intellectual Property Organization* [en línea]. [Consulta: 14 junio 2017]. Disponible en: /designs/es/faq_industrialdesigns.html.

7 Anexos

Tabla 5: HU - Actualización automática de cambios.

Número: 9	**Nombre del requisito:** Actualización automática de cambios.
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir actualización automática de cambios. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para la actualización automática de cambios hay que: - Crear un nuevo documento. - El documento debe contener al menos un objeto. **3- Flujo de la acción a realizar:** El sistema debe permitir la actualización automática de cambios, cuando el usuario modifiqué, cree o elimine un objeto.	
Observaciones:	
Prototipo de interfaz:	

Tabla 6: HU - Exportar tipos de datos estándares

Número: 6	**Nombre del requisito:** Exportar tipos de datos estándares (step, iges, brep)
Programador: Ernesto Alejando Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A

Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir exportar tipos de datos estándares (step, iges, brep). **2- Acciones para lograr el objetivo (precondiciones y datos):** Para exportar tipos de datos estándares hay que: - Debe existir al menos un archivo en formato step, iges y brep. **3- Flujo de la acción a realizar:** Cuando el usuario selecciona la opción exportar, el sistema muestra una ventana de dialogo, donde el usuario indicará el nombre y la ubicación de los archivos a exportar.	
Observaciones:	
Prototipo de interfaz:	

Tabla 7: HU - Visualizar entidades geométricas

Número: 2	**Nombre del requisito:** Visualizar entidades geométricas.
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir visualizar entidades geométricas. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para visualizar entidades geométricas hay que: - Crear un nuevo documento. - Crear objetos dentro del documento. - El objeto debe contener un Shape. **3- Flujo de la acción a realizar:**	

- El sistema debe permitir exportar los shapes de los objetos, esta acción puede realizarse seleccionando la opción Guardar en la interfaz. - El usuario elegirá el nombre del archivo y la dirección donde se guardará. - Si selecciona la opción Cancelar regresará a la vista previa.
Observaciones:
Prototipo de interfaz:

Tabla 8: HU - Guardar los datos generados por un proyecto

Número: 3	**Nombre del requisito:** Guardar los datos generados por un proyecto en un único archivo
Programador: Ernesto Alejando Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Media	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir guardar los datos generados por un proyecto en un único archivo. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para restaurar un proyecto hay que: - Crear un nuevo documento. - Deben existir objetos dentro del documento. - Los datos deben estar serializados y los shapes exportados en formato brep. **3- Flujo de la acción a realizar:** - El sistema debe permitir guardar los datos generados por un proyecto en un único archivo, esta acción puede realizarse seleccionando la opción Guardar en la interfaz. - El usuario elegirá el nombre del archivo y la dirección donde se guardará. - Si selecciona la opción Cancelar regresará a la vista previa.	
Observaciones:	
Prototipo de interfaz:	

Tabla 9: Importar tipos de datos estándares.

Número: 5	**Nombre del requisito:** Importar tipos de datos estándares (step, iges, brep)
Programador: Ernesto Alejando Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Media	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir importar tipos de datos estándares (step, iges, brep). **2- Acciones para lograr el objetivo (precondiciones y datos):** Para importar tipos de datos estándares hay que: - Debe existir al menos un archivo en formato step, iges y brep. **3- Flujo de la acción a realizar:** Cuando el usuario selecciona la opción Importar, el sistema muestra una ventana de dialogo, donde el usuario buscará la ubicación de los archivos en formato step, iges y brep.	
Observaciones:	
Prototipo de interfaz:	

Tabla 10: HU - Manipulación de objetos individuales

Número: 12	**Nombre del requisito:** Manipulación de objetos individuales
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción: **1- Objetivo:** Permitir manipulación de objetos individuales. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para manipular los objetos individuales hay que: - Crear un nuevo documento. - Debe exitir al menos un objeto en el documento. **3- Flujo de la acción a realizar:** El sistema debe permitir la utilización de las vistas ortogonal y perspectiva cuando el usuario lo solicite.
Observaciones:
Prototipo de interfaz:

Tabla 11: HU - Modificar propiedades de los objetos

Número: 13	**Nombre del requisito:** Modificar propiedades de los objetos
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir modificar propiedades de los objetos. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para resaltar modificar propiedades de los objetos hay que: - Crear un nuevo documento. - Debe existir al menos un objeto en el documento. **3- Flujo de la acción a realizar:** El sistema debe permitir modificar propiedades de los objetos cuando el usuario lo solicite.	

Observaciones:
Prototipo de interfaz:

Tabla 12: HU - Múltiples visores de un documento.

Número: 8	**Nombre del requisito:** Múltiples visores de un documento.
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir múltiples visores de un documento. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para obtener múltiples visores de un documento hay que: - Crear un nuevo documento. **3- Flujo de la acción a realizar:** El sistema debe permitir múltiples visores de un documento a petición del usuario.	
Observaciones:	
Prototipo de interfaz:	

Tabla 13: HU - Resaltar objetos

Número: 13	**Nombre del requisito:** Resaltar objetos
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción: **1- Objetivo:** Permitir resaltar objetos. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para resaltar objetos hay que: - Crear un nuevo documento. - Debe existir al menos un objeto en el documento. **3- Flujo de la acción a realizar:** El sistema debe permitir el resaltar objetos cuando el usuario posicione el cursor encima.
Observaciones:
Prototipo de interfaz:

Tabla 14: HU - Restaurar un proyecto

Número: 4	**Nombre del requisito:** Restaurar un proyecto
Programador: Ernesto Alejando Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Media	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A
Descripción: **1- Objetivo:** Permitir restaurar un proyecto. **2- Acciones para lograr el objetivo (precondiciones y datos):** Para restaurar un proyecto hay que: - Debe existir al menos un proyecto generado por la aplicación. **3- Flujo de la acción a realizar:** Cuando el usuario selecciona la opción Abrir, el sistema muestra una ventana de dialogo, donde el usuario buscará la ubicación del archivo de proyecto.	

Observaciones:
Prototipo de interfaz:

Número: 14	**Nombre del requisito:** Seleccionar objetos
Programador: Ernesto Alejandro Santana Hidalgo	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1- Objetivo:

Permitir seleccionar objetos.

2- Acciones para lograr el objetivo (precondiciones y datos):

Para seleccionar objetos hay que:

- Crear un nuevo documento.
- Debe existir al menos un objeto en el documento.

3- Flujo de la acción a realizar:

El sistema debe permitir seleccionar objetos cuando el usuario presione el cursor encima.

Observaciones:
Prototipo de interfaz: